PENGUJIAN KINERJA MQTT *BROKER* BERBASIS KONTAINER MENGGUNAKAN DOCKER PADA PERANGKAT RASPBERRY PI

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Andika Kurniawan
NIM: 135150200111042

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2021

PENGESAHAN

PENGUJIAN KINERJA MQTT *BROKER* BERBASIS KONTAINER MENGGUNAKAN DOCKER PADA PERANGKAT RASPBERRY PI

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Andika Kurniawan
NIM: 135150200111042

Skripsi ini telah diuji dan dinyatakan lulus pada
13 Agustus 2021

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Mahendra Data, S.Kom., M.Kom.	Dany Primanita Kartikasari, S.T., M.Kom.
NIK: 2015038611171001	NIP: 19771116 200501 2 003

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., MMG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 13 Agustus 2021

Andika Kurniawan

NIM: 135150200111042

PRAKATA

Puji Syukur kehadirat Allah SWT yang telah memberikan rahmat, taufik dan hidayah-Nya, sehingga laporan skripsi yang berjudul "PENGUJIAN KINERJA MQTT *BROKER* BERBASIS KONTAINER MENGGUNAKAN DOCKER PADA PERANGKAT RASPBERRY PI" ini dapat terselesaikan.

Penulisan menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Mahendra Data, S.Kom., M.Kom., selaku dosen pembimbing I dan Dany Primanita Kartikasari, S.T., M.Kom, selaku dosen pembimbing II yang telah bersedia untuk membimbing dan memberikan arahan dalam penelitian ini.
2. Adhitya Bhawiyuga, S.Kom., M.Sc., selaku Ketua Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya.
3. Achmad Basuki, S.T., M.MG., Ph.D., selaku Ketua Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya.
4. Ir. Primantara Hari Trisnawan, M.Sc., selaku Sekretaris Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya.
5. Bapak Agi Putra Kharisma, S.T., M.T., selaku dosen Penasihat Akademik.
6. Segenap civitas akademik Fakultas Ilmu Komputer Universitas Brawijaya yang telah memberikan arahan dan ilmu selama Penulis menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
7. Kedua orang tua khususnya Almarhumah Ibu yang telah memberikan segalanya untuk penulis dan seluruh keluarga besar atas segala nasehat, kasih sayang dan kesabaran dalam membesarkan dan mendidik penulis, serta yang senantiasa tiada henti-hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini.
8. Teman-teman seperjuangan Fakultas Ilmu Komputer.

Penulis menyadari bahwa dalam penulisan skripsi ini masih terdapat banyak kekurangan. Namun penulis berharap skripsi ini bermanfaat bagi semua pihak yang menggunakan.

Malang, 13 Agustus 2021

Penulis

andikakurniawan95@gmail.com

ABSTRAK

Andika Kurniawan, Pengujian Kinerja MQTT *Broker* Berbasis Kontainer Menggunakan Docker Pada Perangkat Raspberry Pi.

Dosen Pembimbing: Mahendra Data, S.Kom., M.Kom., dan Dany Primanita Kartikasari, S.T., M.Kom.

Untuk menguji dan mengetahui kinerja MQTT *broker* menggunakan teknologi kontainerisasi pada perangkat Raspberry Pi dengan dua basis *image* yang berbeda dilakukan dengan beberapa parameter pengujian, yaitu pengujian *resource usage*(*cpu* dan *memory*) dengan *script* python yang penulis buat, pengujian *latency* dengan MQTT *broker latency measure tool,* dan pengujian *packet loss* dengan MQTT *broker latency measure tool.* Basis *image* yang digunakan sebagai media pemasangan MQTT *broker* (Mosquitto) adalah belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine. Docker *image* dibuat dari Dockerfile. *Image* dibangun dan dijalankan dengan perintah yang disediakan Docker. Dalam hal *resource usage* (cpu) MQTT *broker* yang berjalan diatas docker dengan basis *image* belenalib/raspberry-pi2 mampu lebih baik dari *native* dengan 0,53% sampai 10,38% penggunaan cpu lebih rendah. Namun dalam hal *resource usage(memory)* kedua MQTT *broker* yang berjalan diatas docker masih lebih tinggi dibandingkan dengan MQTT *broker* yang dijalankan secara *native,* dengan perbedaan lebih dari 50%. Pada pengujian *latency* kedua MQTT *broker* yang berjalan diatas docker masih juga lebih tinggi lebih dari 50% dibandingkan dengan MQTT *broker* yang dijalankan secara *native.* Pada pengujian *packet loss*, MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 memiliki kinerja lebih baik dengan hanya mengalami *packet loss* sebesar 28.60% pada klien MQTT berjumlah 1000 dan QoS 0.

Kata kunci: Raspberry Pi, *Internet of things,* MQTT *broker*, Docker.

ABSTRACT

Andika Kurniawan, *Performance Testing of Container-Based MQTT Broker Using Docker on Raspberry Pi Device*

Dosen Pembimbing: Mahendra Data, S.Kom., M.Kom., dan Dany Primanita Kartikasari, S.T., M.Kom.

To test the performance of the MQTT broker using containerization technology on Raspberry Pi devices with two different image bases, we use several experimental parameters. The experimental parameters are resource usage (CPU and memory), latency, and packet loss. To test the resource usage, we created a program using Python. Meanwhile, to test latency and packet loss, we used the MQTT broker measure latency measure. The base images used as the installation media for the MQTT broker (Mosquitto) are belenalib/raspberry-pi2 and belenalib/raspberry-pi2-alpine. We created the Docker images using Dockerfiles. The image is built and run with the commands provided by the Docker program. Regarding resource usage (CPU), the MQTT broker running on docker based on the belenalib/raspberry-pi2 image performs better than native with 0.53% to 10.38% lower CPU usage. However, the resource usage (memory) of both MQTT brokers running on docker is still higher than MQTT brokers running natively, with a difference of more than 50%. In the second latency test, the MQTT broker that runs on docker is more than 50% higher than the MQTT broker that runs natively. In packet loss testing, MQTT broker with image base belenalib/raspberry-pi2 has better performance with only 28.60% packet loss on 1000 MQTT clients and 0 QoS.

***Keywords**: Raspberry Pi, Internet of things, MQTT broker, Docker.*

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
PRAKATA iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xi
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 2
1.3 Tujuan 2
1.4 Manfaat 3
1.5 Batasan masalah 3
1.6 Sistematika pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Kajian Pustaka 5
2.2 *Internet of Things* (IoT) 7
2.3 MQTT 7
2.3.1 *Publish/Subcribe* 9
2.3.2 Topik MQTT 9
2.3.3 MQTT *Quality of Service* (MQTT QoS) 10
2.3.4 MQTT *Payload* 10
2.3.5 MQTT *Broker* 11
2.3.6 Mosquitto 11
2.4 Virtualisasi 11
2.4.1 Kontainer (*Container*) 12
2.5 Docker 12

2.5.1 Docker *Image* 13
2.5.2 Docker *Container* 13
2.5.3 Dockerfile 13
2.6 Raspberry Pi 14
2.7 *Resource Usage* 14
2.8 *Latency* 15
2.9 *Packet Loss* 15
BAB 3 METODOLOGI 16
3.1 Studi Literatur 16
3.2 Persiapan *Testbed* 17
3.2.1 Perangkat Keras 17
3.2.2 Perangkat Lunak 17
3.3 Topologi pengujian 18
3.4 Alur Menjalankan MQTT *Broker* 20
3.4.1 Dockerfile MQTT *Broker* Belenalib/raspberry-pi2 21
3.4.2 Dockerfile MQTT *Broker* Belenalib/raspberry-pi2:alpine 22
3.5 Pengujian 23
3.5.1 Pengujian *Resource Usage* 23
3.5.2 Pengujian *Latency* 24
3.5.3 Pengujian *Packet Loss* 25
3.6 Pengambilan Data 25
3.7 Pembahasan 26
3.8 Kesimpulan 26
BAB 4 PENGUJIAN DAN HASIL PENGUJIAN 27
4.1 Persiapan Pengujian 27
4.1.1 Dockerfile 28
4.1.1.1 Dockerfile MQTT *Broker* belenalib/raspberry-pi2 28
4.1.1.2 Dockerfile MQTT *Broker* belenalib/raspberry-pi2-alpine 29
4.1.2 Docker *Image* 30
4.1.3 Docker *Container* 31

4.2 Pengujian 32
4.2.1 Pengujian *Resource Usage* 32
4.2.1.1 Pengujian CPU *Usage* 32
4.2.1.2 Pengujian *Memory Usage* 33
4.2.2 Pengujian *Latency* 33
4.2.3 Pengujian *Packet Loss* 34
BAB 5 PEMBAHASAN 36
5.1 Kinerja *Resource Usage* 36
5.1.1 CPU *Usage* 36
5.1.2 *Memory Usage* 37
5.2 Kinerja MQTT *broker* terhadap *Latency* 38
5.3 Kinerja MQTT *broker* terhadap *Packet Loss* 39
BAB 6 Penutup 40
6.1 Kesimpulan 40
6.2 Saran 41
DAFTAR REFERENSI 42

DAFTAR TABEL

Tabel 2.1 Kajian Pustaka 5
Tabel 2.2 Tipe Pesan MQTT (OASIS, 2014) 8
Tabel 3.1 Kebutuhan Perangkat Keras 17
Tabel 3.2 Kebutuhan Perangkat Lunak 18
Tabel 3.3 Pengujian *Resource Usage* 24
Tabel 3.4 Pengujian *Latency* 24
Tabel 3.5 Pengujian *Packet Loss* 25
Tabel 3.6 *Script* Python Resource Usage 26
Tabel 4.1 Langkah Pemasangan Docker di Raspberry Pi 27
Tabel 4.2 Rata-rata Hasil Pengujian CPU *usage* 33
Tabel 4.3 Rata-rata Hasil Pengujian *Memory usage* 33
Tabel 4.4 Rata-rata Hasil Pengujian *Latency* 34
Tabel 4.5 Hasil Pengujian *packet loss* 34

DAFTAR GAMBAR

Gambar 2.1 Arsitektur *Publish/Subscribe* 9
Gambar 2.2 *Virtual Machine* vs *container* (Chelladhurai dkk, 2017) 12
Gambar 2.3 Arsitektur Docker (Docker, -) 13
Gambar 2.4 Raspberry Pi 2 Model B (Raspberry Pi Foundation, -) 14
Gambar 3.1 Tahap Penelitian 16
Gambar 3.2 Topologi Pengujian 19
Gambar 3.3 Alur Data Pengujian 19
Gambar 3.4 Arsitektur MQTT *Broker* 20
Gambar 3.5 Alur menjalankan MQTT *broker* 21
Gambar 3.6 Diagram Alir Dockerfile MQTT *Broker* rpi-raspbian 22
Gambar 3.7 Diagram Alir Dockerfile MQTT *Broker* rpi-alpine 23
Gambar 4.1 Perintah Docker --version 28
Gambar 4.2 Docker build rpidmosq-rasbian 30
Gambar 4.3 Docker build rpidmosq-alpine 30
Gambar 4.4 Docker image ls 30
Gambar 4.5 Docker run rpidmosq-raspbian 31
Gambar 4.6 Docker container ls rpidmosq-ras 31
Gambar 4.7 Docker run rpidmosq-alpine 31
Gambar 4.8 Docker container ls rpidmosq-alp 32
Gambar 5.1 Grafik Pengujian *CPU Usage* 36
Gambar 5.2 Grafik Pengujian *Memory Usage* 37
Gambar 5.3 Grafik pengujian *latency* 38
Gambar 5.4 Grafik pengujian *packet loss* 39

BAB 1 PENDAHULUAN

1.1 Latar belakang

Pemanfaatan internet tidak hanya sebatas digunakan untuk *browsing*, *streaming* atau mengunduh berkas saja, tetapi dapat digunakan untuk hal-hal yang bermanfaat lainnya. Pemanfaatan internet juga tidak terbatas pada perangkat komputer atau *smartphone* saja, bahkan setiap benda seperti lampu dan sensor suhu disekitar rumah dapat terhubung dengan jaringan internet. Hal itu merupakan penerapan dari sebuah konsep yang disebut *Internet of Things* (IoT) (Miller, 2017).

Terdapat berbagai macam protokol di IoT yang dapat digunakan, salah satunya yaitu MQ *Telemetry Transport* (MQTT). MQTT adalah sebuah protokol perpesanan yang menggunakan arsitektur *publish/subscribe* yang sangat simple dan ringan. MQTT seringkali digunakan karena efisien dalam mentransfer data dan penggunaan sumber daya (Mqtt.org, -). Ada dua komponen penting agar MQTT dapat bekerja, yaitu MQTT *broker* dan klien MQTT. MQTT *broker* bertugas untuk menangani dan mengatur semua komunikasi antara klien MQTT. Sedangkan Klien MQTT bertugas untuk melakukan *publish/subscribe* ke MQTT *broker* (Fysarakis dkk., 2016). MQTT *broker* dapat dipasang pada bermacam perangkat dan beberapa sistem operasi, baik perangkat fisik atau perangkat virtual (*Virtual Machine* dan Kontainer).

Seringkali MQTT *broker* seperti Mosquitto, dipasang secara *native* pada perangkat IoT Raspberry Pi. Raspberry Pi dipilih sebagai perangkat IoT yang cocok karena memiliki ukuran yang kecil dan tidak memakan tempat. Mosquitto yang dipasang secara native memiliki keterbatasan seperti tidak tersedianya fitur *clustering*, namun hal ini dapat diatasi dengan memasang Mosquitto pada teknologi *Container* yang disebut Docker.

Container adalah lingkungan virtualisasi terisolasi yang mencakup *dependency (library)* tertentu untuk menjalankan suatu aplikasi (Docker, 2016). Penggunaan sumber daya memori dan penyimpanan pada *container* jauh lebih rendah dari pada VM tradisional (Linux Academy, 2017), sehingga dapat dipasang dan berjalan dengan baik pada Raspberry Pi yang memiliki sumberdaya memori dan penyimpanan yang sangat terbatas. Terdapat banyak teknologi *container* seperti Docker, LXC, dan OpenVZ. Namun, yang akan digunakan pada penelitian ini adalah Docker.

Docker digunakan karena memiliki beberapa keuntungan dibandingkan dengan menggunakan teknologi *container* lain diantaranya yaitu hemat sumberdaya dan fleksibilitas yang tinggi. Docker *container* dibuat menggunakan basis *image* yang tersedia di Docker *registry*. Docker *image* dapat berisi hanya inti dari sebuah Sistem Operasi ataupun aplikasi terpasang yang siap untuk dijalankan.

Membuat *image* baru di Docker terbilang cukup mudah dibandingkan dengan teknologi *container* yang lain. Untuk membuat *image* baru di Docker dapat dilakukan menggunakan Dockerfile, yaitu baris perintah yang disediakan oleh Docker yang akan memasang aplikasi dan *library* di basis *image* yang telah didefinisikan sebelumnya. Kemudian Dockerfile dapat dijalankan untuk membuat *image* baru dengan perintah yang telah disediakan oleh Docker (Bernstein, 2014).

Karena belum adanya pengujian secara spesifik terhadap basis *image* untuk MQTT *broker* yang berjalan di Raspberry Pi, maka penulis melakukan penelitian "Pengujian Kinerja MQTT *Broker* Berbasis Kontainer Menggunakan Docker Pada Perangkat Raspberry Pi" ini untuk mengetahui bagaimana kinerja MQTT *broker* di perangkat Raspberry Pi walaupun menggunakan.

Pada penelitian ini akan dilakukan pengujian terhadap dua Docker *image* MQTT *broker* dengan basis *image* yang sesuai dan dapat berjalan tanpa kendala di Raspberry Pi. Basis image yang akan digunakan yaitu belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine. Selain itu, akan dibandingkan dengan MQTT *broker* yang dipasang secara *native* di raspberry Pi untuk mengetahui perbedaan kinerja MQTT *broker* yang berjalan diatas Docker dan tanpa Docker.

1.2 Rumusan masalah

Berdasar pada latar belakang yang telah dijabarkan di atas, maka dapat diambil rumusan masalah sebagai berikut:

1. Bagaimana pengujian kinerja MQTT *Broker* berbasis kontainer menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine pada perangkat Raspberry Pi ?
2. Bagaimana hasil perbandingan kinerja MQTT *Broker* menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine dan tanpa menggunakan Docker pada perangkat Raspberry Pi?

1.3 Tujuan

Berdasarkan dari rumusan masalah yang telah disebutkan, maka tujuan dari penelitian ini yaitu:

1. Menguji dan mengetahui kinerja MQTT *Broker* berbasis container menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine pada perangkat Raspberry Pi.
2. Mengetahui hasil perbandingan kinerja MQTT *Broker* menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine dan tanpa menggunakan Docker pada perangkat Raspberry Pi.

1.4 Manfaat

Manfaat yang diharapkan dari penelitian ini yaitu:

1. Mengetahui basis *image* Docker untuk MQTT *Broker* pada perangkat Raspberry Pi yang memiliki kinerja terbaik.
2. Hasil pengembangan dapat digunakan untuk penelitian IoT selanjutnya.

1.5 Batasan masalah

Batasan masalah dari penelitian ini agar penelitian yang dilakukan lebih fokus yaitu:

1. Tidak dijelaskan cara instalasi sistem operasi, dan perangkat lunak yang digunakan.
2. Pengujian *latency* dan *packet loss* dilakukan menggunakan *software* MQTT *broker latency measure tool.*

1.6 Sistematika pembahasan

Sistematika pembahasan pada laporan penelitian ini adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini membahas latar belakang masalah yang diangkat peneliti kemudian menentukan rumusan masalah yang selanjutnya digunakan untuk menentukan tujuan dan manfaat dari penelitian. Selanjutnya peneliti juga membahas batasan masalah dan terakhir membahas sistematika pembahasan terkait penelitian dan penulisan laporan.

BAB II LANDASAN KEPUSTAKAAN

Bab ini berisi tinjauan pustaka dari penelitian sebelumnya dan dasar teori pendukung yang berkaitan dengan penelitian ini seperti *Internet of Things* (IoT), MQTT, Virtualisasi, *Container*, dan Docker untuk membantu dalam penelitian.

BAB III METODOLOGI

Dalam bab ini berisi langkah – langkah sistematis dan terstruktur serta metode untuk melakukan penelitian agar dapat berjalan dengan lancar. Dan membahas mengenai metodologi yang digunakan mulai dari studi literatur, perancangan dan pengujian yang digunakan oleh peneliti.

BAB IV PENGUJIAN DAN HASIL

Bab ini berisi hasil dari langkah-langkah yang dilakukan sebelum melakukan pengujian dan ketika pengujian sedang dilakukan selain itu juga menampilkan data yang didapatkan dari pengujian.

BAB V PEMBAHASAN

Bab ini berisi grafik dan penjelasan hasil pengujian yang telah diperolah dari pengujian yang dilakukan.

BAB VI PENUTUP

Berisi tentang penarikan kesimpulan dari hasil penelitian yang telah diolah, kesimpulan yang diambil berdasarkan pada rumusan masalah penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan ini berisi mengenai teori, konsep, dan juga metode, atau sistem dari literatur ilmiah, yang berkaitan dengan penelitian pengujian kinerja MQTT *Broker* berbasis kontainer menggunakan Docker pada perangkat Raspberry Pi ini.

2.1 Kajian Pustaka

Berikut ini beberapa penelitian yang berkaitan dan memiliki kesamaan baik dari segi objek maupun metode dengan penelitian pengembangan MQTT *Broker* berbasis *Operating System-Level Virtualization* menggunakan Docker pada perangkat Raspberry Pi.

Tabel 2.1 Kajian Pustaka

Penulis	Tahun	Judul	Penelitian	Perbedaan
Przyłucki, S., Czerwiński, D., dan Sierszeń, A.	2017	*A performance evaluation of docker-based MQTT server implementation on internet of things device*	Menguji peforma dari implementasi penyedia MQTT pada perangkat Raspberry Pi. Penelitian ini menbandingkan peforma penyedia MQTT pada Raspberry Pi tanpa layer virtualisasi dengan penyedia MQTT pada Raspberry Pi berbasis kontainer Docker. Pengujian dilakukan menggunakan *tool* mqtt-malaria. Penelitian ini hanya melakukan pengujian skalabilitas pada -masing-masing QoS. Hasil yang didapat dari penelitian ini	Penelitian ini melakukan pengujian lalu membandingkan MQTT *broker* yang berjalan di atas Docker (menggunakan dua basis *image* yang berbeda) dengan MQTT *broker* tanpa Docker pada Raspberry Pi. Pengujian yang dilakukan yaitu pengujian *resource usage* menggunakan *script* python, *latency* dan *packet loss* menggunakan MQTT *broker latency measure tools.*

			yaitu pada pengujian pengiriman pesan yang sedikit (1000 pesan perdetik), kinerja MQTT *broker* yang dikontainerisasi mengikuti kinerja MQTT *broker* yang diimplementasikan secara *native*. Kenaikan jumlah pesan perdetik, mempengaruhi penurunan stabilitas pemrosesan pesan. Namun, secara umum perbedaan kinerja antara MQTT *broker* tidak terlalu signifikan, sehingga Docker kontainer dapat dijadikan *platform* yang layak untuk diimplemetasikan pada banyak komponen sistem IoT.	
Morabito, R.	2016	*A performance evaluation of container technologies on Internet of Things device*	Menguji Peforma teknologi kontainer (Docker) pada Raspberry Pi. Pada penelitian ini pengujian dilakukan menggunakan *tool* sysbench, dan yang diuji adalah *CPU, Disk I/O, Network I/O,* dan *Memory*. Selain menggunakan	Penelitian ini melakukan pengujian lalu membandingkan MQTT *broker* yang berjalan di atas Docker (menggunakan dua basis *image* yang berbeda) dengan MQTT *broker* tanpa Docker pada

			Tool, pengujian juga dilakukan menggunakan aplikasi MySQL dan Apache2. Hasil yang didapatkan dari penelitian ini yaitu perbedaan kinerja yang hampir dapat diabaikan antara *container* dengan *native.* Hal ini dikarenakan keduanya memiliki perbedaan kinerja yang sangat tipis.	Raspberry Pi. Pengujian yang dilakukan yaitu pengujian *resource usage* menggunakan *script* python, *latency* dan *packet loss* menggunakan MQTT *broker latency measure tools.*

2.2 *Internet of Things* (IoT)

Internet of Things dikembangkan untuk lebih memperluas manfaat yang didapat dengan adanya jaringan komputer. *Internet of Things* sendiri dapat didefinisikan sebagai objek atau benda yang dapat saling bertukar data memanfaatkan jaringan internet. Menurut Loukides dan Bruner (2015) *Internet of Things* adalah pendekatan untuk menyatukan antara *software* dan *hardware* yang terhubung dengan internet, mengenalkan kecerdasan buatan untuk benda-benda yang sebelumnya tidak cerdas, dan menambahkan *endpoint* fisik untuk *software.*

Istilah *Internet of Things* dipublikasikan secara resmi pada tahun 2005 oleh *International Telecommunication Union* (ITU). Sedangkan pencetus awal istilah Internet of Things sendiri adalah Kevin Ashton (2009) pada tahun 1999. Dengan adanya *Internet of Things* banyak manfaat yang dapat diperoleh salah satu contohnya dari jarah jauh kita dapat memonitoring perubahan yang terjadi pada lingkungan secara *real time* menggunakan internet (Suresh dkk. 2014).

2.3 MQTT

Message Queue Telemetry Transport (MQTT) adalah sebuah protokol transportasi perpesanan *Client Server* dengan metode *publish/subscribe.* MQTT merupakan protokol yang ringan, terbuka, simpel, dan dirancang agar mudah diimplementasikan. Karakteristik seperti ini lah yang membuatnya sangat cocok untuk digunakan di berbagai situasi yang berbeda, termasuk lingkungan yang terbatas seperti komunikasi *Machine to machine* (M2M) dan *Internet of Things* (IoT). Protokol MQTT berjalan diatas layer TCP/IP yang menjamin sampainya data (OASIS, 2014).

Beberapa fitur dari MQTT yaitu:

1. Menggunakan pola pesan *publish/subcribe* yang menyediakan distribusi pesan *one-to-many* dan memisahkan aplikasi.
2. Tidak peduli terhadap isi dari pesan yang dikirimkan.
3. Memiliki tiga level *Quality of Service* (QoS) untuk pengiriman pesan.
4. Transportasi tambahan yang kecil dan memperkecil pertukaran protokol untuk mengurangi lalulintas jaringan.
5. Mekanisme yang memberitahu ketika ada koneksi yang tidak normal terjadi.

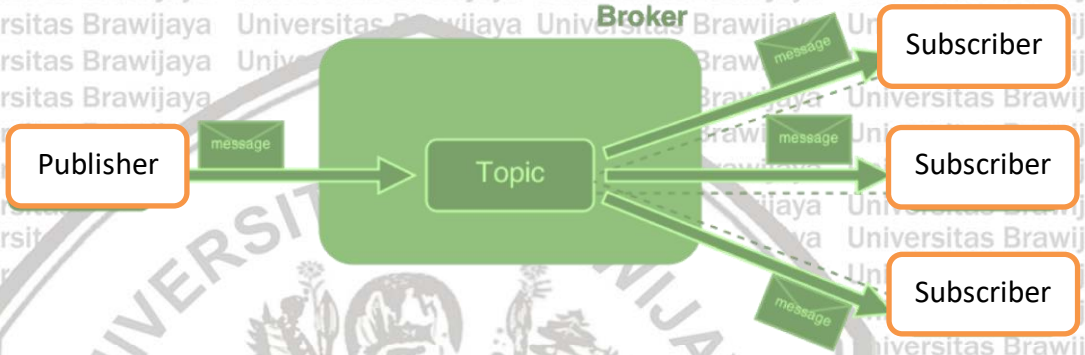
MQTT memiliki 14 tipe pesan perintah atau *Control Packet* yang digunakan saat berkomunikasi antara *Client*(*Publisher/Subscriber*) dengan MQTT *broker*, tipe-tipe pesan perintah ini dapat dilihat pada tabel di bawah ini.

Tabel 2.2 Tipe Pesan MQTT (OASIS, 2014)

Control Packet	**Deskripsi**
CONNECT	*Client request to connect to server*
CONNACK	*Connect Acknowledgment*
PUBLISH	*Publish Message*
PUBACK	*Publish Acknowledgment*
PUBREC	*Publish Received-assured delivery part 1*
PUBREL	*Publish Release-assured delivery part 2*
PUBCOMP	*Publish Completed-assured delivery part 1*
SUBSCRIBE	*Client Subscribe request*
SUBACK	*Subscribe Acknowledgment*
UNSUBSCRIBE	*Client Unsubscribe request*
UNSUBACK	*Unsubscribe Acknowledgment*
PINGREQ	*PING Request*
PINGRESP	*PING Response*
DISCONNECT	*Client is Disconnecting*

2.3.1 *Publish/Subcribe*

G. Coulouris dkk (2012) mengambil kesimpulan bahwa arsitektur *Publish Subcribe* melakukan komunikasi antar entitas lain secara tidak langsung dengan menggunakan perantara. Pada arsitektur ini terdapat tiga komponen utama yang dijadikan sebagai dasar pembentukan arsitektur *Publish/Subcribe* yaitu *broker*, *publisher* dan *subcriber*. Setiap komponen mempunyai tugas masing masing yang saling terhubung satu sama lainnya. *Broker* bertugas untuk menjadi perantara pertukaran data antara *Publisher* dan *Subcriber. Publisher* bertugas untuk mengirim data. *Subcriber* bertugas untuk menerima data. Arsitektur *Publish/Subcribe* dapat dilihat pada Gambar 2.1.



Gambar 2.1 Arsitektur *Publish/Subscribe*

2.3.2 Topik MQTT

Pada MQTT, pengiriman data yang dilakukan oleh *publisher* ke broker, lalu menuju ke *subcriber* didasarkan pada topik. Topik ini dapat memiliki susunan yang bertingkat-tingkat (hirarki). Setiap level topik dipisahkan oleh pemisah *level-topic* (/). Pemisah *level-topic* digunakan untuk memperkenalkan struktur pada nama topik. Sebuah penyaring langganan topik dapat berisi karakter *wilcard*, yang memungkinkan untuk *subscribe* beberapa topik sekaligus.

Wilcard pada topik MQTT dibagi menjadi dua yaitu *Multi-level* dan *Single level wilcard*. *Multi-level wilcard* yang ditandai dengan '#' adalah karakter *wilcad* yang cocok dengan semua level topik, *Multi-level wilcard* mewakili topik *parent* dan semua level *child* atatu topik setelahnya dari topik tersebut, misalnya suhu/rumah/# artinya *subcriber* akan menerima data dari suhu/rumah, suhu/rumah/halaman, dan. suhu/rumah/halaman/depan. *Single level wilcard* yang ditandai dengan '+' adalah karakter *wilcard* yang cocok hanya dengan satu level topik, misalnya suhu/rumah/+, artinya *subcriber* akan menerima data suhu/rumah/halaman, suhu/rumah/kolam, dan suhu/rumah/dapur (OASIS, 2014).

2.3.3 MQTT *Quality of Service* (MQTT QoS)

MQTT memiliki 3 level *Quality of Service* (QoS) untuk mengirim pesan yaitu QoS 0, QoS 1, dan QoS 2 (OASIS, 2014).

1. QoS 0 (*at most once*): Didasarkan pada upaya terbaik atau tercepat dalam pengiriman data, tetapi tidak mempedulikan apakah data tersebut sampai. Kekurangan dari level QoS ini yaitu dapat terjadi kehilangan data.
2. QoS 1 (*at least once*): Data dikirimkan dan dipastikan sampai, tapi duplikasi dapat terjadi. Jika *broker* tidak menerima ACK dari *subscriber*, broker akan mengirim ulang data.
3. QoS 2 (*exactly once*): Data dikirimkan dan dipastikan tersampaikan (tidak terjadi duplikasi), level QoS ini adalah QoS yang paling aman, namun dapat terjadi *overhead*, dan lebih lambat dibanding dengan QoS level lain.

2.3.4 MQTT *Payload*

Payload dapat berupa data tambahan atau muatan yang dapat dikirimkan pada suatu jaringan melalui prokol jaringan tertentu. Beberapa tipe pesan perintah dari MQTT yang memiliki *payload* adalah sebagai berikut (OASIS, 2014):

1. CONNECT

   Pada CONNECT, *Payload* berisi satu atau lebih *string* yang disandikan dengan format UTF-8. Mereka menentukan *unqiue-identifier* untuk klien, *Will-topic* dan pesan, serta Nama Pengguna dan Kata Sandi yang akan digunakan. Semuanya kecuali yang pertama adalah opsional dan ada tidaknya ditentukan berdasarkan *flag* di *header variabel*.

2. SUBSCRIBE

   Payload pada SUBSCRIBE berisi daftar nama topik di mana klien dapat berlangganan, dan tingkat QoS. String ini dikodekan dengan format UTF.

3. SUBACK

   Pada SUBACK, *Payload* berisi daftar level QoS yang diberikan. Ini adalah tingkat QoS di mana administrator untuk *server* memberika izin pada klien untuk berlangganan topik tertentu. Level QoS yang diberikan terdaftar dalam urutan yang sama dengan topik dalam pesan SUBSCRIBE yang sesuai.

4. PUBLISH

   Payload pada PUBLISH hanya dapat berisi data khusus aplikasi. Tidak dapat berisi data yang dibuat secara natural dan bagian pada pesan diperlakukan sebagai tipe data BLOB.

2.3.5 MQTT *Broker*

MQTT menggunakan arsitektur *publish-subscribe,* oleh karena itu, MQTT *Broker* adalah elemen yang harus ada pada protokol MQTT. MQTT *Broker* menjadi perantara antara *publisher* dan *subscriber* dalam mengirim data. *Publisher* tidak akan dapat mengirim data dan *subscriber* tidak dapat melakukan *subcribe* dan menerima data jika tidak ada MQTT *broker* (HiveMQ, 2015).

Berdasarkan pada artikel yang ditulis oleh HiveMQ (2015) MQTT *broker* memiliki tanggung jawab untuk menerima semua pesan dari *publisher*, memfilter pesan yang diterima berdasarkan topik, menentukan *subscriber* yang berlangganan, dan mengirimkan pesan ke *subscriber* yang berlangganan. MQTT *broker* juga dapat diatur untuk melakukan otentikasi, otorisasi, dan integrasi klien. Hal ini dilakukan karena MQTT *broker* seringkali diekspos secara langsung di internet dan menangani banyak klien. Oleh karenanya faktor keamanan juga diperhatikan. Salah satu MQTT *broker* yang populer yaitu Mosquitto.

2.3.6 Mosquitto

Mosquitto mampu menyediakan server dan klien dari implemetasi protokol pesan MQTT yang memenuhi standar. Mosquitto dapat digunakan untuk semua situasi yang membutuhkan perpesanan ringan, apalagi dengan perangkat yang memiliki sumber daya yang terbatas. Proyek Mosquitto merupakan bagian dari Eclipse Foundation. Terdapat tiga bagian pada proyek ini yaitu, server mosquitto, klien mosquitto_pub dan mosquitto_sub, dan *library* klien MQTT yang ditulis dalam bahasa C (Light, 2017).

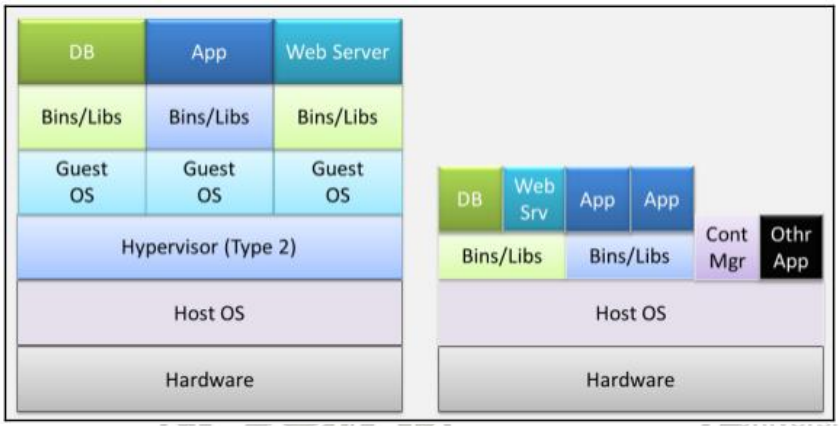
2.4 Virtualisasi

Virtualisasi adalah teknologi yang memungkinkan host fisik menjalankan sistem operasi lain secara virtual. Ini menambahkan lapisan virtualisasi antara sistem operasi dan perangkat keras fisik yang dirancang berdasarkan arsitektur x86 (Vemulapalli dan Mada, 2014). Semua sistem terdiri dari unit kontrol, penyimpanan, aritmatika dan unit logika (ALU), perangkat input dan output. Lapisan virtualisasi menangani beberapa host virtual oleh berbagi memori secara dinamis, Device I/O dan CPU yang terbuat dari ALU dan unit kontrol untuk mengakses perangkat keras (Bunyakitanon dan Peng, 2014). Virtualisasi menyederhanakan pengembangan perangkat lunak dengan memungkinkan abstraksi perangkat keras dan konsolidasi *server* yang membantu mengganti perangkat keras fisik tunggal ke beberapa sistem virtual (Vemulapalli dan Mada, 2014). Dengan demikian, mengurangi biaya, ruang fisik, dan energi. Arsitektur virtualisasi bisa berupa arsitektur *host* atau arsitektur *hypervisor*. Arsitektur *host* menjalankan lapisan virtualisasi sebagai aplikasi di atas sistem operasi sedangkan arsitektur *hypervisor* menjalankan lapisan virtualisasi secara langsung pada sistem. Arsitektur berbasis *hypervisor* lebih efisien daripada *host* karena tidak berada pada

sistem operasi dan memiliki akses langsung ke perangkat keras, yang meningkatkan kinerja (VMWare, 2007).

2.4.1 Kontainer (*Container*)

Container adalah lingkungan virtual terisolasi pada tingkat Sistem Operasi yang mencakup seperangkat dependensi tertentu untuk menjalankan aplikasi tertentu. Linux *Container* memiliki pendekatan yang berbeda dari *hypervisor*. Linux *container* sangat ringan karena tidak mem-virtualisasikan perangkat keras namun *container* pada suatu *host* fisik hanya menggunakan satu *host kernel* secara efisien. Tidak seperti *Virtual Machine* (VM) yang menjalankan OS secara penuh, sebuah *container* dapat berisi satu proses saja (Chelladhurai dkk, 2017). Perbedaan struktur dari VM dan *container* dapat dilihat pada Gambar 2.2 berikut ini.



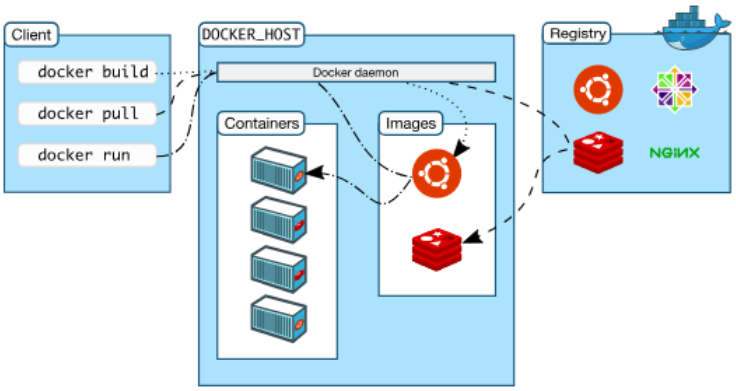
Gambar 2.2 *Virtual Machine* vs *container* (Chelladhurai dkk, 2017)

2.5 Docker

Docker merupakan salah satu teknologi *containerization* yang populer. Docker diciptakan oleh Docker Inc. Docker merupakan proyek *open source* sehingga komunitas Docker dapat bekerja dan berbagi untuk memperbaiki teknologi ini sehingga menguntungkan untuk semua pihak baik pengguna, komunitas, maupun Docker Inc. Perusahaan, Docker Inc., juga membangun karya komunitas Docker, membuatnya lebih aman, dan membagikan kemajuan tersebut kembali ke komunitas yang lebih besar. Dengan Docker, pengguna dapat memperlakukan kontainer seperti mesin virtual modular yang sangat ringan. Dan mendapatkan fleksibilitas dengan kontainer tersebut. Docker *image* dapat dibuat, disebarkan, disalin, dan dipindahkannya dari lingkungan satu ke lingkungan yang lain (Docker, -).

Docker menggunakan arsitektur *client-server*. Docker *client* memberitahukan kepada Docker *daemon* yang melakukan tugas yang cukup berat dalam membangun, menjalankan, dan mendistribusikan Docker *container.* Docker

client dan Docker *daemon* dapat berjalan pada sistem yang sama, atau bisa juga menghubungkan Docker *client* ke Docker *daemon* secara remote. Docker *client* dan *daemon* dapat berkomunikasi dengan sebuah REST API melalui socket UNIX atau sebuah antarmuka jaringan (Docker, -). Gambar 2.3 dibawah ini, merupakan ilustrasi dari arsitektur Docker.



Gambar 2.3 Arsitektur Docker (Docker, -)

2.5.1 Docker *Image*

Docker memiliki sebuah Docker *image template* yang hanya dapat dibaca (*read-only*) dengan perintah untuk membuat Docker *container.* Sering kali sebuah *image* berbasis pada *image* lain dengan beberapa tambahan. Sebagai contoh membangun sebuah *image* menggunakan basis ubuntu tetapi ditambah ftp *server* dan beberapa aplikasi lain di dalamnya (Docker, -).

2.5.2 Docker *Container*

Docker *container* adalah *image* yang telah dijalankan. *Container* dapat dibuat, dijalankan, dihentikan, dipindahkan, atau dihapus dengan menggunakan Docker API ataupun CLI. Sebuah *container* dihubungkan dengan satu atau lebih jaringan, menambahkan penyimpanan, atau bahkan membuat *image* baru dari *container* yang ada. Secara *default,* sebuah *container* terisolasi dari *container* lainnya dan *host*. Tapi, isolasi *container* ini, dapat dikontrol baik pada jaringan, penyimpanan, atau *subsistem* yang mendasari dari *container* lain atau dari *host* (Docker, -).

2.5.3 Dockerfile

Dockerfile adalah baris perintah yang digunakan untuk membuat Docker *image*. Setiap perintah pada Dockerfile membuat layer pada sebuah basis *image* yang definisikan sebelumnya. Ketika membuat Dockerfile dengan beberapa perintah dan membangunnya, hanya perintah itu yang berubah pada *image* yang baru. Dengan Dockerfile pada Docker membuat *image* adalah sebuah pekerjaan sangat mudah, ringan, dan cepat dibandingkan dengan teknologi virtualisasi lain (Docker, -).

2.6 Raspberry Pi

Raspberry Pi adalah sebuah komputer mini atau sering disebut sebagai *single-board computer*. Raspberry Pi didesain untuk mengajarkan anak muda untuk membuat program, dan Raspberry Pi dapat digunakan untuk melakukan berbagai macam hal. Raspberry Pi dapat menjalankan beberapa sistem operasi seperti linux, ataupun windows core. Raspberry tidak memiliki penyimpanan bawaan sehingga dibutuhkan sebuah Micro SD card untuk menjalan sistem operasi dan menyimpan *file*. Raspberry Pi menerima daya dari *micro-usb* yang terhubung pada *power adaptor* atau pada berbagai perangkat yang memiliki konektivitas USB. Sama seperti komputer pada umumnya, Raspberry Pi dapat menjadi komputer konvensional dengan hanya menancapkan *mouse*, *keyboard* dan monitor di *port* yang telah disediakan. Di banyak penelitian, Raspberry Pi digunakan sebagai perangkat IoT karena sangat *powerfull* seperti komputer konvensional, tetapi menggunakan daya yang sedikit dan ukurannya yang kecil. Raspberry Pi diciptakan oleh sebuah perusahaan yang bernama Raspberry Pi Foundation, perusahaan ini berbasis di Inggris (Raspberry Pi Foundation, -). Raspberry Pi memiliki beberapa jenis perangkat yang memiliki spesifikasi yang berbeda, salah satu jenis Raspberry Pi yang cukup populer yaitu Raspberry Pi 2 Model B yang ditunjukkan oleh Gambar 2.4.

Gambar 2.4 Raspberry Pi 2 Model B (Raspberry Pi Foundation, -)

2.7 *Resource Usage*

Setiap sistem komputer pasti menggunakan sumber daya yang jumlahnya terbatas yang digunakan pada waktu tertentu. Sumber daya yang digunakan pada sistem komputer yaitu CPU (*processor*), *storage* (penyimpanan), dan *memory* (RAM) (Nicholson, 2014). Setiap sumber daya memiliki fungsi yang berbeda, dan akan mengalami gangguan ketika penggunaannya terlampau tinggi.

CPU bertugas untuk menangani setiap proses dan perintah yang diminta oleh sistem operasi dan perangkat lunak yang dijalankan oleh pengguna komputer. Ketika penggunaan CPU tinggi, maka sebuah proses baru harus menunggu proses sebelumnya selesai, dan hal ini menyebabkan keterlambatan proses. *Storage* bertugas sebagai tempat menyimpan data yang disimpan oleh sistem operasi, perangkat lunak maupun pengguna komputer. Ketika *storage* penuh maka tidak akan dapat digunakan untuk menyimpan data baru. *Memory* bertugas untuk menyimpan data sementara dari proses dan perintah yang dilakukan oleh CPU. Saat penggunaan *memory* tinggi dan mencapai batasnya, maka sebuah proses baru akan menunggu hingga proses sebelumnya selesai atau bahkan tidak dapat melakukan proses baru sama sekali.

2.8 *Latency*

Latency mengacu pada waktu yang dibutuhkan oleh sebuah *packet* data untuk dipindahkan dari *host* sumber ke *host* tujuan (Armitage dkk, 2006). Kecepatan pengiriman data pada suatu jaringan dapat diketahui berdasarkan nilai *latency.* Ketika nilai *latency* rendah, maka kecepatan pengiriman data tinggi. Namun, ketika nilai *latency* tinggi, maka kecepatan pengiriman data rendah.

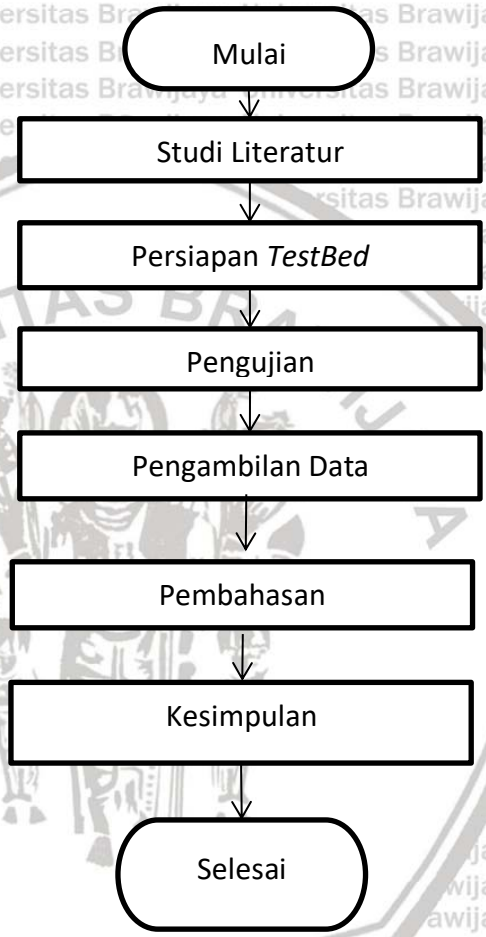
2.9 *Packet Loss*

Packet loss dapat diketahui ketika suatu *packet* data tidak mencapai *host* tujuan. Hal ini dapat terjadi ketika *packet* data hilang di suatu jaringan. *Packet loss* sering digambarkan dengan rasio jumlah *packet* data yang hilang per jumlah *packet* data yang dikirim (Armitage dkk, 2006). *Packet loss* dapat diartikan sebagai persentase *packet* data yang hilang pada sebuah transportasi jaringan (Thulin, 2004). Beberapa penyebab *packet loss* yaitu (Armitage dkk, 2006):

- Terlalu banyak *packet* data yang dikirimkan
- Layer *link* yang error sehingga menyebabkan *packet corrupt.*

BAB 3 METODOLOGI

Di bab ini menjelaskan metode, teknik, atau langkah-langkah yang digunakan dalam penelitian pengujian kinerja mqtt *broker* berbasis kontainer menggunakan docker pada perangkat raspberry pi. Dalam bab ini akan dijelaskan tahapan metode yang digunakan untuk mengerjakan penelitian, analisis kebutuhan, metode pengujian, analisa hasil pengujian dan kesimpulan. Tahapan penelitian dapat dilihat pada gambar 3.1.



Gambar 3.1 Tahap Penelitian

3.1 Studi Literatur

Dalam penelitian ini, studi literatur di perlukan untuk menunjang penulisan serta pengerjaan penelitian. Studi literatur merupakan dasar teori secara detail, yang dibahas pada bab 2. Dalam penyusunan dasar teori, referensi tentang teori-teori yang diperlukan didapatkan dari artikel, buku, jurnal, *website* terpercaya maupun penelitian-penelitian terkait. Studi literatur nantinya dijadikan pedoman pengetahuan dasar dalam melakukan tahapan-tahapan penelitian. Teori yang mendukung penelitian ini adalah sebagai berikut:

1. *Internet of Things (IoT)*
2. MQTT
3. Virtualisasi
4. Docker
5. Raspberry Pi
6. *Latency*
7. *Packet Loss*
8. *Resource Usage*

3.2 Persiapan *Testbed*

Persiapan *testbed* dilakukan untuk menyiapkan bahan yang diperlukan dan mendukung seluruh tahapan dalam penelitian. Dua hal yang perlu dipersipakan dalam penilitian ini yaitu perangkat lunak dan perangkat keras.

3.2.1 Perangkat Keras

Table 3.1 adalah daftar perangkat keras yang dibutuhkan untuk melakukan pengujian MQTT *broker* dalam penelitian.

Tabel 3.1 Kebutuhan Perangkat Keras

Perangkat Keras	Spesifikasi	Keterangan
Raspberrry Pi 2	-Chipset Boardcom BCM2836 Soc -Quad-Core ARM Cortex-A7 -MicroSD 16 GB -RAM 1 GB	Sebagai perangkat untuk menjalankan MQTT *broker* menggunakan Docker.
Asus A455LD Core i5	-Processor Intel Core i5-4210U -RAM 8 GB -Haddisk 1TB (500 GB + 500 GB)	Untuk mengakses Raspberry Pi, dan sebagai klien *subscriber* dan *subcriber* MQTT.

3.2.2 Perangkat Lunak

Table 3.2 adalah daftar perangkat lunak yang dibutuhkan untuk melakukan pengujian MQTT *broker* dalam penelitian ini.

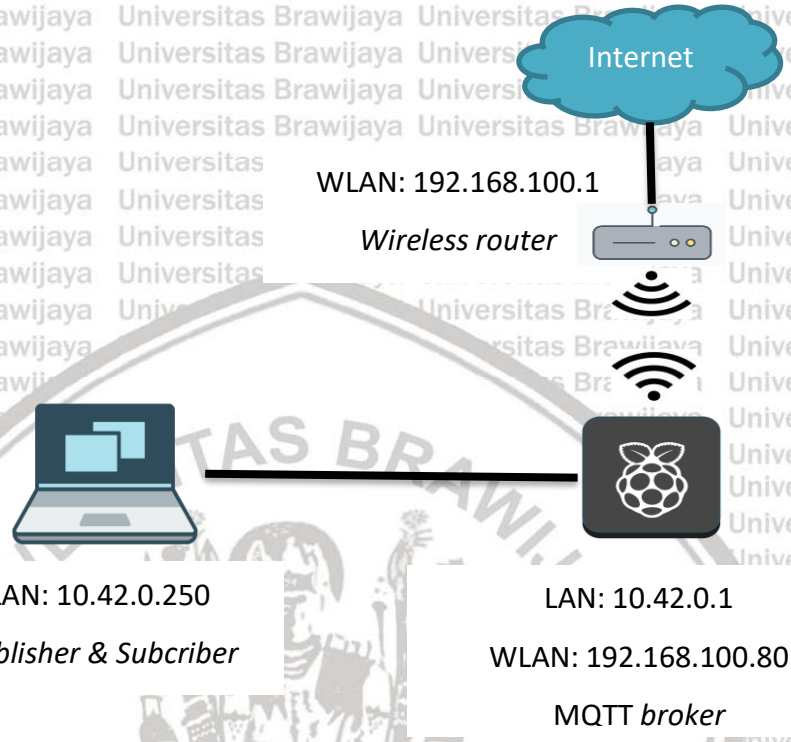
Tabel 3.2 Kebutuhan Perangkat Lunak

Perangkat Keras	Perangkat Lunak	Keterangan
Asus A455LD Core i5	Ubuntu 16.04 LTS 64 bit	Sistem operasi pada Asus A455LD Core i5.
	Visual Studio Code 1.30	IDE pemrograman untuk membuat *script* Dockerfile di Raspberry Pi.
	Go go1.11.1 linux/amd64	Bahasa pemrograman untuk menjalankan MQTT *broker latency measure tool*
	MQTT *broker latency measure tool* 1.0	*Tools* untuk pengujian *latency* dan *packet loss*
Raspberrry Pi 2	Raspbian Jessie 2017-02-16	Sistem operasi pada Raspberry Pi 2.
	Docker 18.06.1-ce	Teknologi container.
	Mosquitto 1.5.3	Aplikasi MQTT *broker.*
	Python 2.7	Bahasa pemrograman untuk klien MQTT.
	Paho MQTT	Library Python untuk Klien MQTT.
	Belenalib/raspberry-pi2	Basis *image* Docker untuk MQTT *broker.*
	Belenalib/raspberry-pi2-alpine	Basis *image* Docker untuk MQTT *broker.*

3.3 Topologi pengujian

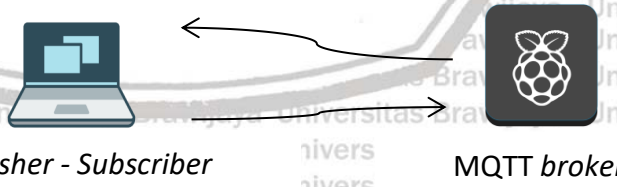
Pengujian dilakukan di lingkungan lokal penguji. Dimana terdapat perangkat Laptop Asus A455LD yang terhubung dengan Raspberry Pi melalui jaringan LAN dengan kabel, jaringan LAN ini disediakan oleh Raspberry Pi yang diatur sebagai akses point. Klien MQTT (*Publisher* dan *Subscriber* dengan bantuan MQTT *latency tools* sebagai tools pengujian) berada pada perangkat Asus A455LD dengan alamat IP 10.42.0.250. Karena *publisher* dan *subscriber* berada pada

perangkat yang sama, maka memiliki alamat IP yang sama pula. Sedangkan MQTT *broker* berada pada Raspberry Pi dengan alamat IP 10.42.0.1. Raspberry Pi juga terhubung dengan *Wireless router* secara *wireless* agar dapat terhububung dengan internet. Topologi lingkungan pengujian dapat dilihat pada gambar 3.2.



Gambar 3.2 Topologi Pengujian

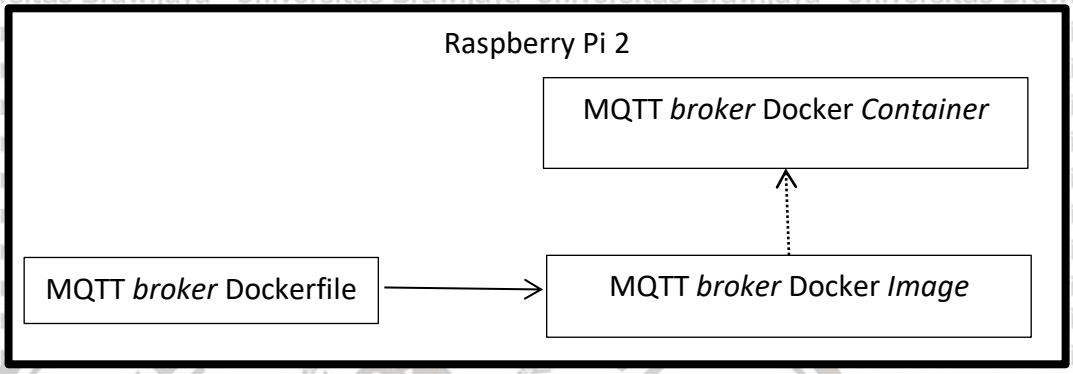
MQTT *latency tools* pada perangkat laptop Asus A455LD akan menjalan *sampler* MQTT *publisher* yang mengirimkan data *dummy* ke MQTT *broker* di Raspberry Pi, kemudian data *dummy* yang diterima MQTT *broker* akan diteruskan ke *sampler* MQTT *subscriber*. Seperti yang terlihat pada alur data pengujian pada gambar 3.3.



Gambar 3.3 Alur Data Pengujian

3.4 Alur Menjalankan MQTT *Broker*

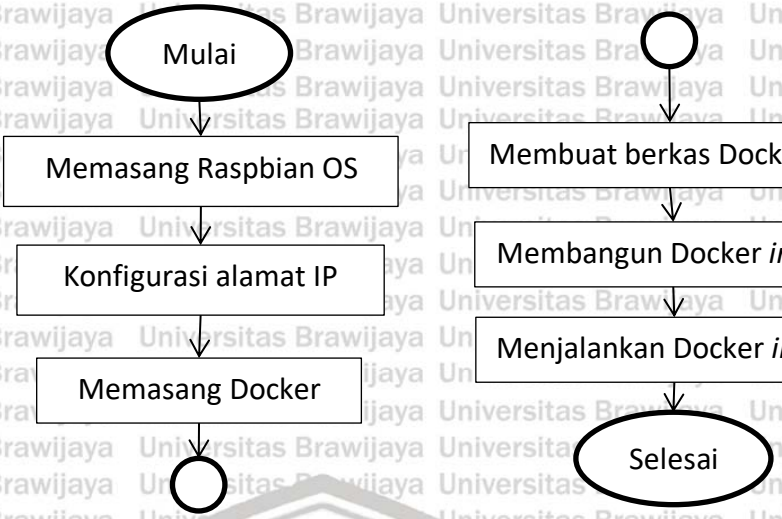
Arsitektur MQTT *broker* yang akan diuji dapat dilihat pada Gambar 3.4. Dimana akan diawali dengan membuat dua buah Dockerfile yang berisi MQTT *broker*(Mosquitto) dengan basis *image* Belenalib/raspberry-pi2 dan Belenalib/raspberry-pi2-alpine yang kemudian akan di-*generate* menjadi Docker *image*. Dua Docker *image* MQTT *broker* inilah yang nantinya akan dijalankan menjadi container berisi MQTT *broker* di Raspberry Pi dan menjadi objek pengujian.



Gambar 3.4 Arsitektur MQTT *Broker*

Sebelum pengujian dapat dimulai, hal yang perlu dilakukan adalah melakukan pemasangan Raspbian OS di Raspberry Pi. Proses pemasangan dapat dilakukan dengan mengunduh NOOBS pada website resmi Raspberry Pi. Lalu menyalin hasil ekstraksi *directory* NOOBS ke Micro SD Card. Kemudian melakukan booting dan memasang Raspbian OS di Raspberry Pi tersebut. Langkah selanjutnya yaitu melakukan konfigurasi alamat IP di Laptop dan Raspberry Pi, agar laptop dan Raspberry Pi dapat terhubung dengan jaringan LAN. Langkah ketiga melakukan pemasangan Docker di Raspberry Pi.

Langkah keempat, membuat file Dockerfile yang berisi perintah pemasangan MQTT *broker* Mosquitto dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine agar image yang dibuat dapat dijalankan di Raspberry Pi. Langkah kelima membangun Docker *image* dari Dockerfile yang telah dibuat dengan perintah yang telah disediakan oleh Docker. Yang terakhir menjalankan Docker *image* MQTT *broker* yang telah dibangun. Gambar 3.5 berikut ini merupakan diagram alur dari menjalankan MQTT *broker* yang akan diuji.

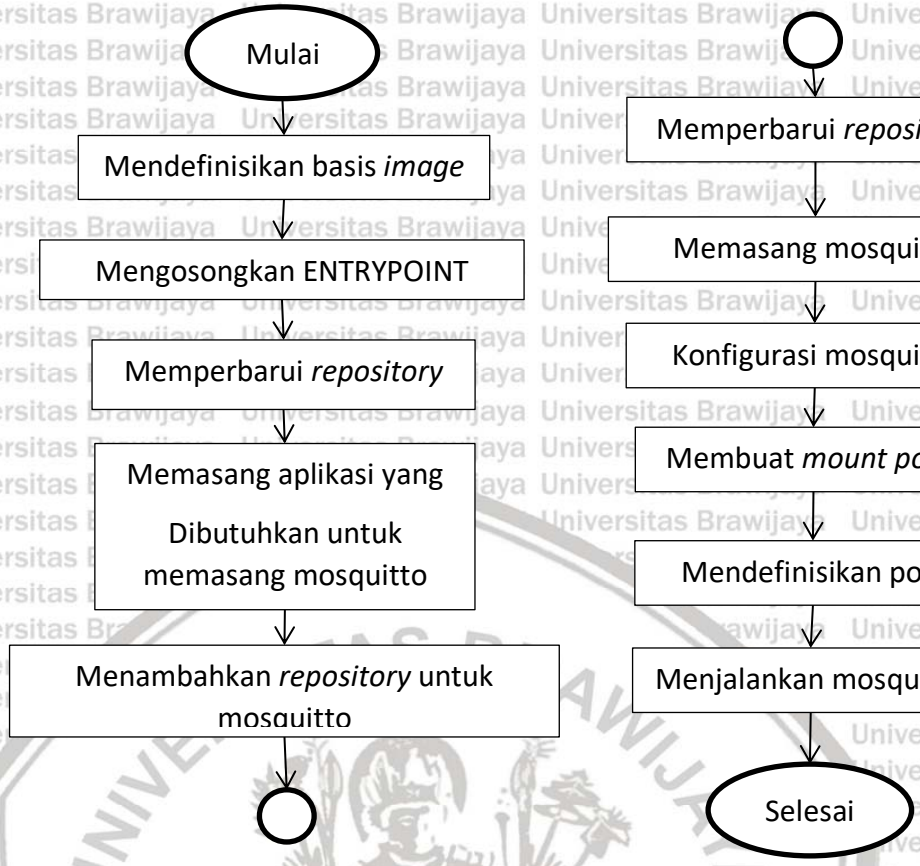


Gambar 3.5 Alur menjalankan MQTT *broker*

3.4.1 Dockerfile MQTT *Broker* Belenalib/raspberry-pi2

Hal pertama yang harus dilakukan dalam pembuatan dockerfile ini adalah mendefinisikan basis *image* yang digunakan. Basis *image* yang digunakan yaitu belenalib/raspberry-pi2. Langkah kedua mengosongkan ENTRYPOINT, hal ini dilakukan karena *image* belenalib/raspberry-pi2 memiliki perintah bawaan yang otomatis berjalan ketika *image* dijalankan. Langkah ketiga, memperbarui daftar aplikasi dari *repository.* Langkah ke-empat, memasang aplikasi yang dibutuhkan untuk memasang Mosquitto. Langkah kelima, menambahkan sumber *repository* untuk memasang mosquitto, dan Langkah ke-enam melakukan pemasangan Mosquitto.

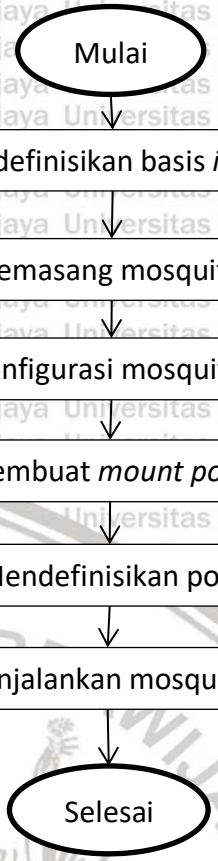
Langkah ketujuh, melakukan konfigurasi Mosquitto dengan membuat *directory* mosquitto yang didalamnya tedapat tiga *directory* yaitu config, data, dan log. Lalu, menyalin file konfigurasi mosquitto.conf ke *directory* config. Kemudian, mengganti kepemilikan dan hak akses *directory* mosquitto. Langkah kedelapan, membuat *mountpoint* untuk *directory* config, data, dan log. Langkah kesembilan, mendefinisikan *port* yang dibuka yaitu *port* 1883. Dan langkah terakhir menjalankan Mosquitto berdasarkan pada file konfigurasi mosquitto.conf. Gambar 3.6 berikut ini merupakan diagram alir Dockerfile MQTT *Broker* menggunakan basis *image* Belenalib/raspberry-pi2.



Gambar 3.6 Diagram Alir Dockerfile MQTT *Broker* rpi-raspbian

3.4.2 Dockerfile MQTT *Broker* Belenalib/raspberry-pi2:alpine

Hal pertama yang dilakukan dalam pembuatan Dockerfile ini yaitu mendefinisikan basis *image* belenalib/raspberry-pi2:alpine. Langkah kedua, melakukan pemasangan Mosquitto. Langkah ketiga, melakukan konfigurasi Mosquitto dengan membuat *directory* mosquitto, yang didalamnya tedapat tiga *directory* lain yaitu config, data, dan log. Lalu, menyalin file konfigurasi mosquitto.conf ke *directory* config. Kemudian, mengganti kepemilikan dan hak akses *directory* mosquitto. Langkah ke-empat, membuat *mountpoint* untuk *directory* config, data, dan log. Langkah kelima, mendefinisikan port yang dibuka yaitu *port* 1883. langkah terakhir mejalankan Mosquitto berdasarkan pada file konfigurasi mosquitto.conf. Gambar 3.7 berikut ini merupakan diagram alir Dockerfile MQTT Broker menggunakan basis *image* Belenalib/raspberry-pi2:alpine.



Gambar 3.7 Diagram Alir Dockerfile MQTT *Broker* rpi-alpine

3.5 Pengujian

Pada tahap ini, hal yang dilakukan adalah pengujian. Pengujian adalah metode yang peneliti lakukan untuk mendapat hasil nilai data kinerja MQTT *broker*. Pada penilitian ini terdapat tiga variable yang peneliti fokuskan, yaitu *Resource Usage*, *Latency* dan *Packet Loss*. Terdapat tiga MQTT *broker* yang akan diuji di penelitian ini, yaitu MQTT *broker* yang dibangun menggunakan *image* berbasis belenalib/raspberry-pi2, MQTT *broker* yang dibangun menggunakan *image* belenalib/raspberry-pi2:alpine, kedua MQTT broker ini dijalan diatas docker *container*, dan MQTT *broker* yang dijalankan tanpa docker (*native*).

Pengujian dalam penelitian ini, dibagi menjadi tiga. Pengujian pertama yaitu pengujian *resource usage*(CPU dan *memory usage*), pengujian kedua yaitu pengujian *latency,* dan pengujian ketiga yaitu pengujian *packet loss.*

3.5.1 Pengujian *Resource Usage*

Pengujian *Resource Usage* dilakukan untuk mengetahui beban penggunaan CPU, dan penggunaan *memory*. Pengujian akan dilakukan dengan menggunakan klien sebanyak 200, 400, 600, 800, dan 1000 pada MQTT *broker* yang menggunakan basis *image* belenalib/raspberry-pi2, MQTT *broker* yang

menggunakan basis *image* belenalib/raspberry-pi2-alpine, dan MQTT *broker* yang dijalankan tanpa Docker (*Native*). Hasil pengujian akan disimpan pada tabel 3.3.

Tabel 3.3 Pengujian *Resource Usage*

Jumlah Klien	Raspbian	Alpine	*Native*
200			
400			
600			
800			
1000			

Kedua parameter *resource usage*, akan menggunkan struktur tabel yang sama hanya dibedakan isi dan satuannya, dimana *CPU usage* dengan persen (%) sedangkan *memory usage* dengan *Mega Byte* (MB)

3.5.2 Pengujian *Latency*

Pengujian *latency* dilakukan untuk mengetahui berapa lama waktu yang dibutuhkan untuk mengirim data dari *publisher* ke *subscriber*. Satuan yang digunakan oleh *latency* adalah detik (s). Pengujian akan dilakukan dengan menggunakan klien sebanyak 200, 400, 600, 800, dan 1000 pada MQTT *broker* yang menggunakan basis *image* belenalib/raspberry-pi2, MQTT *broker* yang menggunakan basis *image* belenalib/raspberry-pi2-alpine, dan MQTT *broker* yang dijalankan tanpa Docker (*Native*). Pengujian dilakukan menggunakan MQTT *broker* *latency measure* tool yang dibuat oleh Jian Hui. Hasil pengujian akan dimasukkan dalam tabel 3.4.

Tabel 3.4 Pengujian *Latency*

Jumlah Klien	Raspbian	Alpine	*Native*
200			
400			
600			
800			
1000			

3.5.3 Pengujian *Packet Loss*

Pengujian *packet loss* dilakukan untuk mengetahui berapa persentase *packet* yang hilang pada pengiriman data dari *publisher* ke *subscriber*. Setiap satu skenario jumlah klien MQTT dilakukan tiga kali pengujian, dengan QoS yang berbeda. Pengujian akan dilakukan dengan menggunakan klien sebanyak 200, 400, 600, 800, dan 1000. Selain itu, pengujian juga dilakukan menggunakan tiga QoS berbeda yaitu Qos 0, Qos 1, dan Qos 2 pada MQTT *broker* yang menggunakan basis *image* belenalib/raspberry-pi2, MQTT *broker* yang menggunakan basis *image* belenalib/raspberry-pi2-alpine, dan MQTT *broker* yang dijalankan tanpa Docker (*Native*). Pengujian dilakukan menggunakan MQTT *broker latency measure* tool yang dibuat oleh Jian Hui. Hasil pegujian akan disimpan pada tabel 3.5.

Tabel 3.5 Pengujian *Packet Loss*

	Raspbian			Alpine			*Native*		
Jumlah Klien	R Qos 0	R Qos 1	R Qos 2	A Qos 0	A Qos 1	A Qos 2	N Qos 0	N Qos 1	N Qos 2
200									
400									
600									
800									
1000									

3.6 Pengambilan Data

Pengambilan data dilakukan saat pengujian berlangsung. Pengambilan data dilakukan dengan dua sumber berikut:

1. *Script* Python

   Cara pengambilan data penggunaan CPU dan *memory* yaitu dengan menjalanakan *script* python pada Tabel 3.6 berikut pada Raspberry Pi. Setiap 5 detik, *Script* akan mengambil dan mencatat penggunaan CPU dan *memory* berdasarkan PID aplikasi MQTT *broker* yang bekerja. Dan hasil pencatatan tersebut akan disimpan pada file usg.txt. `p.cpu_percent` merupakan baris perintah untuk mendapatkan persentase penggunaan CPU, dan `(p.memory_percent() * totalmem)/100` merupakan baris perintah untuk mendapatkan persentase penggunaan *memory.*

Tabel 3.6 *Script* Python Resource Usage

```
import psutil
p = psutil.Process(6435)
totalmem = psutil.virtual_memory().total / 1000000
print str(totalmem) + " MB"
print psutil.cpu_count()
print p.name()
for i in range(12):
    cpuusg = str(p.cpu_percent(interval=5))
    memusg = str((p.memory_percent() * totalmem)/100)
    print cpuusg + " %, " + memusg + " MB"
    file = open("usg.txt", "a+")
    file.write(cpuusg + ", " + memusg + "\n")
    file.close
```

2. MQTT *broker latency measure tool*

MQTT *broker latency measure tool* menangkap dan menamplikan data *Latency* dan *Packet Loss.*

3.7 Pembahasan

Ketika data pengujian telah didapatkan, data akan diolah dan diubah kedalam grafik dan akan dilakukan analisa untuk mendapatkan hasil kinerja MQTT *broker* yang diuji. Kemudian membandingkan kedua MQTT *broker* yang dijalankan di atas docker dengan dua basis *image* yang berbeda dan MQTT *broker* yang dijalankan secara *native*.

3.8 Kesimpulan

Setelah semua tahap penelitian mulai dari studi literatur sampai dengan mendapatkan hasil dari pengujian selesai dilakukan, selanjutnya dapat dilakukan penarikan kesimpulan berdasarkan penelitian yang telah dilakukan. Kesimpulan menjadi lebih valid dan terpercaya karena didapatkan dari data hasil pengujian, sehingga kesimpulan dapat menjawab rumusan masalah. Kemudian memberikan rekomendasi atau saran untuk penelitian selanjutnya.

BAB 4 PENGUJIAN DAN HASIL PENGUJIAN

Bab ini menjelaskan hal-hal yang dilakukan sebelum dan saat melakukan pengujian MQTT *broker* menggunakan Docker pada Raspberry Pi. Pengujian dilakukan berdasarkan pada yang telah dijelaskan pada bab sebelumnya.

4.1 Persiapan Pengujian

Teknologi *container* yang digunakan pada penelitian ini adalah Docker, lebih jelasnya Docker versi 18.06.1-ce. Docker dipasang pada perangkat Raspberry Pi. Berikut ini langkah-langkah dan penjelasan untuk melakukan pemasangan Docker pada Raspberry Pi.

Tabel 4.1 Langkah Pemasangan Docker di Raspberry Pi

Langkah	Perintah	Deskripsi
1	`sudo apt update`	Memperbarui daftar aplikasi dari *repository*
2	`sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common`	Pemasangan beberapa aplikasi yang dibutuhan untuk menambahkan daftar aplikasi dari *repository* lain
3	`curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -`	Menambahkan kunci GPG resmi dari Docker
4	`echo "deb [arch=armhf] https://download.docker.com/linux/debian $(lsb_release -cs) stable" |sudo tee /etc/apt/sources.list.d/docker.list`	Menambahkan *repository* Docker dengan daftar aplikasi *stable* pada arsitektur armhf
5	`sudo apt update`	Memperbarui daftar aplikasi dari *repository* yang telah ditambahkan
6	`sudo apt install docker-ce`	Memasang Docker

Kemudian untuk dapat mengetahui keberhasilan pemasangan dan versi Docker dapat dilakukan dengan perintah:

```
docker --version
```

Dengan hasil sebagai berikut, hasil dapat berbeda tergantung pada versi Docker yang terpasang.

```
pi@raspberrypi:~ $ docker --version
Docker version 18.06.1-ce, build e68fc7a
```

Gambar 4.1 Perintah Docker --version

Gambar 4.1 menunjukkan versi Docker yang terpasang, yakni Docker versi 18.06.1-ce.

4.1.1 Dockerfile

Dockerfile merupakan baris perintah untuk pemasangan aplikasi, pemasangan *library,* dan konfigurasi pada *image* yang akan digunakan sebagai media MQTT *broker* menggunakan Docker. Terdapat dua implemetasi dockerfile pada penelitian ini, yaitu Dockerfile MQTT Broker menggunakan *image* belenalib/raspberry-pi2 dan Dockerfile MQTT Broker menggunakan *image* belenalib/raspberry-pi2-alpine.

4.1.1.1 Dockerfile MQTT Broker belenalib/raspberry-pi2

Berikut ini merupakan baris perintah Dockerfile untuk membuat *image* MQTT *broker* menggunakan Docker pada Raspberry Pi dengan basis *image* belenalib/raspberry-pi2. Pembuatan Dockerfile dimulai dengan mendefinisikan basis *image* yang digunakan yaitu belenalib/raspberry-pi2 dengan menuliskan pada *script*:

```
FROM belenalib/raspberry-pi2:latest
```

Selanjutnya mengosongkan ENTRYPOINT dengan menuliskan pada *script*:

```
ENTRYPOINT []
```

Hal ini dilakukan karena *image* belenalib/raspberry-pi2 memiliki perintah bawaan yang otomatis berjalan ketika *image* dijalankan. Kemudian memperbarui daftar aplikasi dari *repository* dan memasang aplikasi yang dibutuhkan untuk memasang Mosquitto dengan menuliskan pada *script*:

```
RUN apt update && apt install apt-transport-https -y && apt
install wget -y
```

Selanjutnya menambahkan sumber *repository* untuk memasang mosquitto, dan melakukan pemasangan Mosquitto dengan menuliskan pada *script*:

```
RUN wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
&& apt-key add mosquitto-repo.gpg.key && cd
/etc/apt/sources.list.d/ && wget
http://repo.mosquitto.org/debian/mosquitto-jessie.list && apt
update && apt install mosquitto -y
```

Lalu melakukan konfigurasi Mosquitto dengan membuat *directory* mosquitto yang didalamnya tedapat tiga *directory* yaitu config, data, dan log, lalu menyalin file

konfigurasi mosquitto.conf ke *directory* config. Kemudian mengganti kepemilikan dan hak akses *directory* mosquitto dengan menuliskan pada *script*:

```
RUN mkdir -p /mosquitto/config /mosquitto/data /mosquitto/log &&
cp /etc/mosquitto/mosquitto.conf /mosquitto/config && chown -R
mosquitto:mosquitto /mosquitto
```

Kemudian membuat *mountpoint* untuk *directory* config, data, dan log dengan menuliskan pada *script*:

```
VOLUME ["/mosquitto/config", "/mosquitto/data", "/mosquitto/log"]
```

Selanjutnya mendefinisikan *port* yang dibuka yaitu port 1883 dengan menuliskan pada *script*:

```
EXPOSE 1883
```

Hal terakhirnya yang harus dilakukan yaitu Mejalankan Mosquitto berdasarkan pada file konfigurasi mosquitto.conf dengan menuliskan pada *script*:

```
CMD ["/usr/sbin/mosquitto", "-c", "/mosquitto/config/
mosquitto.conf"]
```

4.1.1.2 Dockerfile MQTT Broker belenalib/raspberry-pi2-alpine

Berikut ini merupakan baris perintah Dockerfile untuk membuat *image* MQTT *broker* menggunakan Docker pada Raspberry Pi dengan basis *image* belenalib/raspberry-pi2-alpine. Pembuatan Dockerfile dimulai dengan mendefinisikan basis *image* yang digunakan yaitu belenalib/raspberry-pi2-alpine dengan menuliskan pada *script*:

```
FROM belenalib/raspberry-pi2-alpine:latest
```

Selanjutnya melakukan pemasangan Mosquitto dengan menuliskan pada *script*:

```
RUN apk --no-cache add mosquitto
```

Kemudian melakukan konfigurasi Mosquitto dengan membuat *directory* mosquitto yang didalamnya tedapat tiga *directory* yaitu config, data, dan log, lalu menyalin file konfigurasi mosquitto.conf ke *directory* config. Kemudian mengganti kepemilikan dan hak akses *directory* mosquitto dengan menuliskan pada *script*:

```
RUN mkdir -p /mosquitto/config /mosquitto/data /mosquitto/log &&
cp /etc/mosquitto/mosquitto.conf /mosquitto/config &&
chown -R mosquitto:mosquitto /mosquitto
```

Lalu membuat *mountpoint* untuk *directory* config, data, dan log dengan menuliskan pada *script*:

```
VOLUME ["/mosquitto/config", "/mosquitto/data", "/mosquitto/log"]
```

Selanjutnya mendefinisikan port yang dibuka yaitu port 1883 dengan menuliskan pada *script*:

```
EXPOSE 1883
```

Hal terakhirnya yang harus dilakukan yaitu Mejalankan Mosquitto berdasarkan pada file konfigurasi mosquitto.conf dengan menuliskan pada *script*:

```
CMD ["/usr/sbin/mosquitto", "-c", "/mosquitto/config/mosquitto.conf"]
```

4.1.2 Docker *Image*

Setelah dockerfile dibuat, selanjutnya yaitu membangun *image* dengan perintah:

```
Docker build -t [repository:tag] [path]
```

Sehingga perintah yang digunakan untuk membangun *image* pada penelitian ini yaitu `Docker build -t rpidmosq-raspbian .` untuk membangun *image* MQTT *broker* berdasarkan basis *image* belenalib/raspberry-pi2. Dan `Docker build -t rpidmosq-alpine .` untuk membangun *image* MQTT *broker* berdasarkan basis *image* belenalib/raspberry-pi2-alpine. Image yang dihasilkan dapat dilihat menggunakan perintah `Docker image ls`.

```
pi@raspberrypi:~/dockerfile/rpidmosq-raspbian $ docker build -t rpidmosq-raspbian .
```

Gambar 4.2 Docker build rpidmosq-rasbian

Gambar 4.2 merupakan perintah untuk membangun *image* rpidmosq-rasbian dari dockerfile yang telah dibuat sebelumnya. Perintah ini dijalankan pada *directory* rpidmosq-raspbian karena dockerfile dibuat pada *directory* tersebut.

```
pi@raspberrypi:~/dockerfile/rpidmosq-alpine $ docker build -t rpidmosq-alpine .
```

Gambar 4.3 Docker build rpidmosq-alpine

Gambar 4.3 merupakan perintah untuk membangun *image* rpidmosq-alpine dari dockerfile yang telah dibuat sebelumnya. Perintah ini dijalankan pada *directory* rpidmosq-alpine karena dockerfile dibuat pada *directory* tersebut.

```
pi@raspberrypi:~ $ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
rpidmosq-raspbian   latest              4afdf8b1f7ba        2 weeks ago
 187MB
rpidmosq-alpine     latest              45860c4e8d1b        2 weeks ago
 21.5MB
```

Gambar 4.4 Docker image ls

Pada gambar 4.4 dapat diketahui bahwa pembangunan *image* rpidmosq-raspbian dan rpidmosq-alpine berhasil dilakukan. Saat membangun sebuah *image* atau melakukan *pull*, Docker secara otomatis men-*generate unique image ID* sehingga setiap *image* memiliki ID yang berbeda. Terdapat perbedaan ukuran *image* pada *image* yang dibangun yaitu 187 MB pada image rpidmosq-rapbian dan 21,5 MB pada image rpidmosq-alpine. Hal ini dapat terjadi karena basis *image* yang digunakan memang berbeda.

4.1.3 Docker *Container*

Hal terakhir yang dilakukan pada persiapan pengujian MQTT *broker* ini yaitu mejalankan *image* yang telah berhasil dibangun. *Image* MQTT *broker* ini dapat dijalankan dengan menggunakan perintah:

```
Docker run [option] image [command]
```

Untuk lebih jelasnya, perintah spesifik yang digunakan yaitu `docker run -tidp 1883:1883 --name="nama_image" repository_image`. Dimana `docker run` merupakan perintah untuk menjalankan *image*. Dengan *option* `-tidp`, `-ti` untuk mengalokasikan *pseudo*-TTY dan membuat *bash interactive*, `-d` agar memungkinkan *container* untuk berjalan di *background,* dan `-p` mem-*publish port* yaitu *port* 1883:1883. *Option* `--name` untuk membuat nama secara spesifik rpidmosq-alp untuk MQTT *broker* bebasis *image* apline dan rpidmosq-ras untuk MQTT *broker* bebasis *image* raspbian. *Container* yang berjalan dapat dilihat menggunakan perintah `docker container ls`.

```
pi@raspberrypi:~ $ docker run -tidp 1883:1883 --name="rpidmosq-ras" rpidmosq-ras
pbian
fde3fcd3617eeaa9480569fbc5a699c642c370b545fff3a91dd14e93494f1aae
```

Gambar 4.5 Docker run rpidmosq-raspbian

Gambar 4.5 menunjukkan perintah yang digunakan untuk menjalankan *image* rpidmosq-raspbian dan perintah ini membuat container dengan nama rpidmosq-ras. Docker men-*generate unique container ID* secara otomatis sehingga setiap *container* memiliki *container ID* yang berbeda. *Container ID* yang diperoleh pada *container* ini yaitu fde3fcd3617eeaa9480569fbc5a699c642c370b545fff3a91dd14e93494f1aae

```
pi@raspberrypi:~ $ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                    NAMES
fde3fcd3617e        rpidmosq-raspbian   "/usr/sbin/mosquitto…"   10 seconds ago
     Up 7 seconds        0.0.0.0:1883->1883/tcp   rpidmosq-ras
```

Gambar 4.6 Docker container ls rpidmosq-ras

Gambar 4.6 menunjukkan perintah untuk melihat daftar *cotainer* yang berjalan. Dapat diketahui bahwa *container* rpidmosq-ras berhasil dijalankan dan memiliki ID yang sama dengan ID yang didapatkan saat menjalankan *image* rpidmosq-raspbian. Diketahui pula *container* berjalan pada *port* 1883.

```
pi@raspberrypi:~ $ docker run -tidp 1883:1883 --name="rpidmosq-alp" rpidmosq-alp
ine
1d9c802671de21d7313e5f27338c67c03ec7de05d7e9f27b1c1ba2fa5c445c38
```

Gambar 4.7 Docker run rpidmosq-alpine

Gambar 4.7 menunjukkan perintah yang digunakan untuk menjalankan *image* rpidmosq-alpine dan perintah ini membuat container dengan nama rpidmosq-alp. *Container ID* yang diperoleh pada *container* ini yaitu 1d9c802671de21d7313e5f27338c67c03ec7de05d7e9f27b1c1ba2fa5c445c38

```
pi@raspberrypi:~ $ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                    NAMES
1d9c802671de        rpidmosq-alpine     "/usr/bin/entry.sh /…"   2 minutes ago
     Up 2 minutes        0.0.0.0:1883->1883/tcp   rpidmosq-alp
```

Gambar 4.8 Docker container ls rpidmosq-alp

Dari Gambar 4.8 dapat diketahui bahwa *container* rpidmosq-alp berhasil dijalankan dan memiliki ID yang sama dengan ID yang didapatkan saat menjalankan *image* rpidmosq-alpine. Selain itu *container* berjalan pada *port* 1883.

4.2 Pengujian

Terdapat tiga MQTT *broker* yang akan diuji di penelitian ini, yaitu MQTT *broker* yang dibangun menggunakan *image* berbasis belenalib/raspberry-pi2, MQTT *broker* yang dibangun menggunakan *image* belenalib/raspberry-pi2-alpine, kedua MQTT broker ini dijalan diatas docker *container*, dan MQTT *broker* yang dijalankan tanpa docker (*native*).

Pengujian dalam penelitian ini, dibagi menjadi tiga. Pengujian pertama yaitu pengujian *resource usage*, pengujian ini dilakukan untuk mengetahui beban raspberry saat MQTT *broker* berjalan. Pengujian kedua yaitu pengujian *latency*, pengujian ini dilakukan untuk mengetahui berapa lama waktu yang dibutuhkan agar data dari *publisher* sampai ke *subscriber*. Pengujian ketiga yaitu pengujian *packet loss*, pengujian ini dilakukan untuk mengetahui berapa persentase *packet* yang hilang dalam pengiriman data dari *publisher* ke *subcriber*, dan mengetahui apakah fitur QoS pada MQTT tetap bekerja.

4.2.1 Pengujian *Resource Usage*

Pengujian *resource usage* dilakukan dengan menjalankan program klien MQTT di ASUS A455LD sebanyak 200, 400, 600. 800. dan 1000 dengan perbandingan 50% *publisher* dan 50% *subscriber*. Data dikirimkan setiap 5 detik sekali.

4.2.1.1 Pengujian *CPU Usage*

Hasil pengujian CPU *usage* didapatkan dengan menjalankan *script* python untuk mendapatkan CPU *usage* MQTT *broker* di Rapberry Pi. *Script* python akan mengambil sample CPU *usage* sebanyak 12 dan mengambil rata-rata dari sample yang didapatkan. Rata-rata hasil pengujian CPU *usage* ditunjukkan oleh tabel 4.2.

Tabel 4.2 Rata-rata Hasil Pengujian CPU *usage*

Jumlah Klien	Raspbian	Alpine	*Native*
200	16.35%	30.68%	16.88%
400	72.36%	93.78%	82.74%
600	99.67%	99.45%	99.11%
800	99.77%	99.65%	99.97%
1000	99.86%	99.87%	100.00%

4.2.1.2 Pengujian *Memory Usage*

Hasil pengujian *memory usage* didapatkan dengan menjalankan *script* python untuk mendapatkan *memory usage* MQTT *broker* di Rapberry Pi. *Script* python akan mengambil sample *memory usage* sebanyak 12 dan mengambil rata-rata dari sample yang didapatkan. Satuan yang digunakan oleh *memory usage* adalah MB(*Mega Byte*). Rata-rata hasil pengujian *memory usage* ditunjukkan oleh tabel 4.3.

Tabel 4.3 Rata-rata Hasil Pengujian *Memory usage*

Jumlah Klien	Raspbian	Alpine	*Native*
200	13.04 MB	10.83 MB	3.90 MB
400	13.15 MB	11.37 MB	4.42 MB
600	14.42 MB	13.82 MB	6.07 MB
800	14.81 MB	14.81 MB	7.24 MB
1000	15.09 MB	16.11 MB	8.18 MB

4.2.2 Pengujian *Latency*

Pengujian *Latency* dilakukan dengan bantuan aplikasi MQTT *broker latency measure tool* yang dibuat oleh Jian Hui pada 2018. *Tool* akan mencatat waktu ketika data akan dipublish, lalu waktu yang didapatkan akan dikirimkan bersamaan dengan data ke MQTT *broker* dengan topik tertentu dengan cara menyisipkannya melalui *payload*. Ketika pesan berhasil diterima oleh *subscriber,* kemudian *tool* akan mencatat waktu saat itu juga dan memecah *payload* yang dikirimkan tadi. Waktu saat data diterima akan kurangi dengan waktu saat data dikirimkan, sehingga didapatkan hasil *latency* yang kemudian disimpan kedalam array. Untuk mendapatkan rata-rata *latency*, *tool* akan memanggil *plugin* yang fungsinya

menjumlahkan semua hasil perhitungan *latency* lalu membaginya dengan banyaknya jumlah data *latency* yang diterima.

Tool ini dijalankan pada ASUS A455LD dengan klien MQTT sebanyak 200, 400, 600. 800. dan 1000 dengan perbandingan 50% *publisher* dan 50% *subscriber*. Data yang dikirimkan oleh setiap *publisher* sebanyak 48 data dan menggunakan QoS 2. Satuan yang digunakan pada *Latency* ini adalah detik (s). Hasil pengujian *latency* ditunjukkan oleh tabel 4.4 berikut ini.

Tabel 4.4 Rata-rata Hasil Pengujian *Latency*

Jumlah Klien	Raspbian	Alpine	*Native*
200	0.16	0.20	0.08
400	0.32	0.39	0.19
600	0.48	0.56	0.30
800	0.67	0.80	0.42
1000	1.04	1.23	0.55

4.2.3 Pengujian *Packet Loss*

Pengujian *Packet Loss* dilakukan dengan bantuan aplikasi MQTT *broker latency measure tool* yang dibuat oleh Jian Hui pada 2018. *Tool* akan menghitung persentase data yang sampai dari total data yang dikirimkan. *Tool* ini dijalankan pada ASUS A455LD dengan klien MQTT sebanyak 200, 400, 600. 800. dan 1000 dengan perbandingan 50% *publisher* dan 50% *subscriber*. Data yang dikirimkan oleh setiap *publisher* sebanyak 48 data. Selain itu, pengujian juga dilakukan dengan 3 pilihan QoS, yaitu Qos 0, QoS 1, dan QoS 2. Hasil pengujian *packet loss* ditunjukkan oleh tabel 4.5 berikut ini.

Tabel 4.5 Hasil Pengujian *packet loss*

	Raspbian			Alpine			*Native*		
Jumlah Klien	R Qos 0	R Qos 1	R Qos 2	A Qos 0	A Qos 1	A Qos 2	N Qos 0	N Qos 1	N Qos 2
200	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
400	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
600	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
800	0.00%	0.00%	0.00%	10.50%	0.00%	0.00%	12.44%	0.00%	0.00%
1000	28.60%	0.00%	0.00%	30.12%	0.00%	0.02%	29.76%	0.00%	0.00%

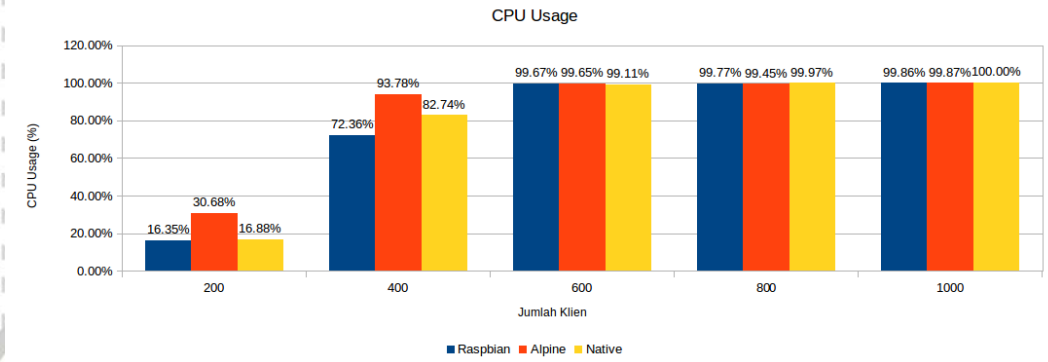
Kode yang berada pada bagian depan QoS merupakan kode untuk setiap MQTT *broker* yang diuji, dimana R untuk MQTT *broker* dengan basis *image* belenalib/raspberry-pi2, A untuk MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine, dan N untuk MQTT *broker* yang dijalankan secara *native* di Raspberry Pi.

BAB 5 PEMBAHASAN

Pada bab pembahasan ini akan dijelaskan dari data hasil pengujian yang telah didapatkan dan diolah menjadi grafik sehingga akan didaptkan bahan untuk penarikan kesimpulan pada bab terkahir.

5.1 Kinerja *Resource Usage*

5.1.1 CPU *Usage*



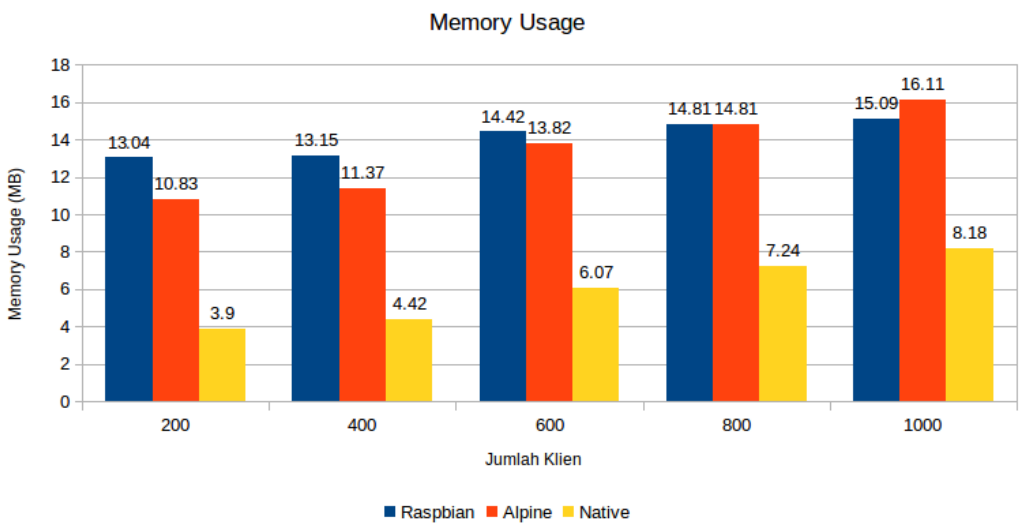
Gambar 5.1 Grafik Pengujian *CPU Usage*

Berdasarkan pada Gambar 5.1 dapat diketahui bahwa meningkatnya CPU *usage* berbanding lurus dengan meningkatnya jumlah klien MQTT. Walapun terjadi peningkatan yang sangat signifikan dari jumlah klien MQTT 200 ke 400, dimana terjadi peningkatan lebih dari 35%.

CPU *usage* MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine pada Raspberry Pi saat jumlah klien MQTT 200 dan 400 lebih tinggi dari pada MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 dan MQTT *broker* yang dijalankan secara *native*. Bahkan saat klien MQTT baru mencapai 400, CPU *usage* belenalib/raspberry-pi2-alpine telah menyentuh 93%. Saat terdapat 600, 800 dan 1000 klien MQTT yang terhubung dan mengirim data, CPU *usage* hampir sama yaitu 99% lebih, namun hanya MQTT *broker* yang dijalankan secara *native* yang menyentuh 100% CPU *usage* dari pengujian ini.

Dapat diambil kesimpulan bahwa dalam hal CPU *usage* MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 lebih rendah dibandingkan dengan belenalib/raspberry-pi2-alpine atau *native* sekalipun.

5.1.2 *Memory Usage*

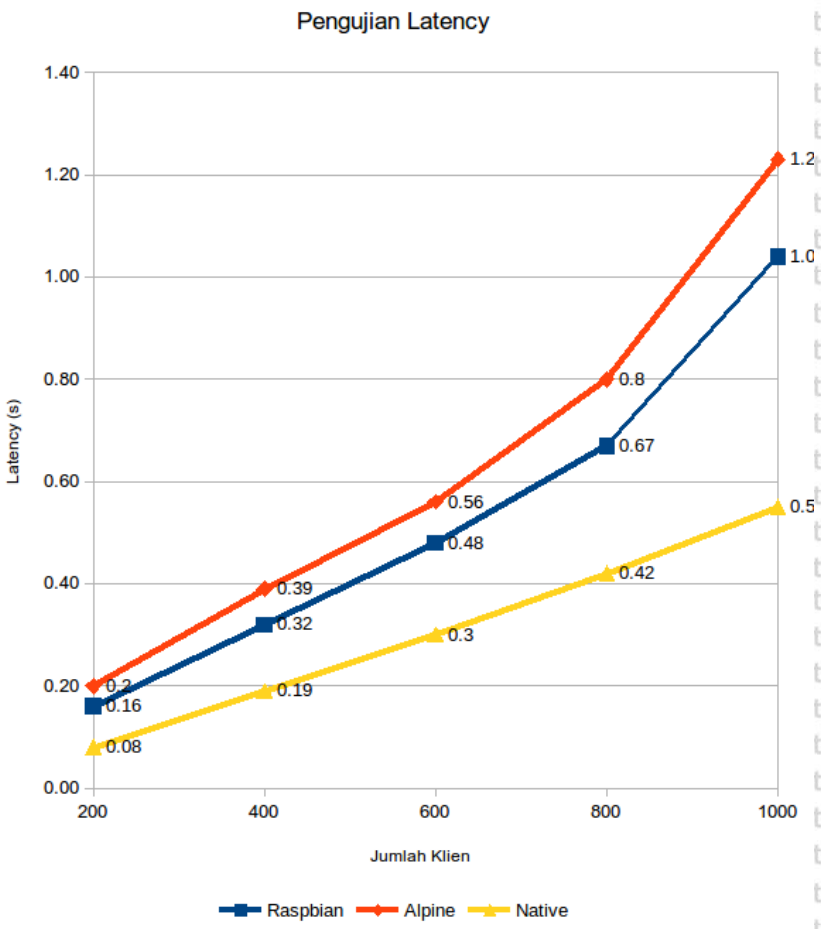


Gambar 5.2 Grafik Pengujian *Memory Usage*

Gambar 5.2 Menunjukkan bahwa peningkatan *Memory usage* berbanding lurus dengan peningkatan jumlah klien. *Memory usage* MQTT *broker* yang dijalankan secara *native* di Raspberry Pi jauh lebih rendah dibandingkan dengan MQTT *broker* yang berjalan diatas Docker baik dengan basis *image* belenalib/raspberry-pi2 maupun belenalib/raspberry-pi2-alpine. Bahkan *memory usage* MQTT *broker* yang dijalankan secara *native* tidak sampai 50% dari *memory usage* kedua MQTT *broker* yang berjalan di atas Docker. Hal ini dapat terjadi karena MQTT *broker* yang dijalankan secara *native* tidak membutuhkan aplikasi lain untuk dijalankan, berbeda dengan kedua MQTT *broker* yang dikembangkan membutuhkan *Docker Container* agar dapat berjalan.

Dapat diambil kesimpulan bahwa dalam hal *memory usage*, MQTT *broker* yang dijalankan secara *native* lebih baik dari MQTT *broker* yang berjalan di atas docker karena *memory usage* nya lebih rendah.

5.2 Kinerja MQTT *broker* terhadap *Latency*

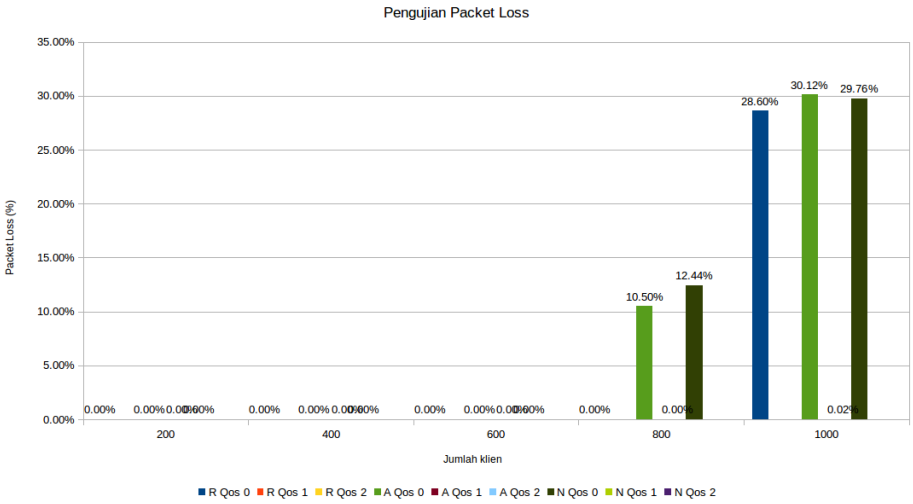


Gambar 5.3 Grafik pengujian *latency*

Berdasarkan gambar 5.3 diketahui bahwa peningkatan *latency* berbanding lurus dengan peningkatan jumlah klien MQTT. Dimana peningkatan *latency* pada MQTT *broker* yang dijalankan secara *native* sangat stabil, namun terjadi peningkatan *latency* yang signifikan pada MQTT *broker* yang dijalankan di atas Docker. Peningkatan yang signifikan ini terjadi saat MQTT klien dari 800 ke 1000, dimana untuk MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine mengalami peningkatan *latency* 0,43 detik, sedangkan MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 mengalami peningkatan *latency* 0,28 detik.

MQTT *broker* yang dijalankan secara *native* meliliki *latency* yang lebih rendah daripada kedua MQTT *broker* yang menggunakan Docker. MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine memiliki latency paling tinggi. Dari sini dapat diambil kesimpulan bahwa MQTT *broker* yang dijalankan secara *native* memiliki peforma yang lebih baik dari MQTT *broker* yang berjalan di atas docker dalam hal *latency*.

5.3 Kinerja MQTT *broker* terhadap *Packet Loss*



Gambar 5.4 Grafik pengujian *packet loss*

Dari Gambar 5.4 menunjukkan bahwa *packet loss* baru dapat terjadi pada klien MQTT berjumlah 800 untuk MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine dan MQTT *broker* yang dijalankan secara *native* pada QoS 0. Untuk MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 *packet loss* baru terjadi pada klien MQTT berjumlah 1000 dan QoS 0 pula. Disini hal yang janggal terjadi dimana MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine mengalami *packet loss* pada klien MQTT berjumlah 1000 dan Qos 2 walaupun nilainya hanya 0.02%. Hal ini mungkin terjadi karena MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine merasa terbebani oleh lalu lintas data yang padat. Selain itu, MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine mengalami persetase *Packet loss* tertinggi yang terjadi pada saat klien MQTT berjumlah 1000 dan QoS 0 yaitu sebesar 30.12%.

Dari pengujian ini, dapat diambil kesimpulan bahwa MQTT *broker* dengan basis *image* belenalib/raspberry-pi2 memiliki peforma yang lebih baik dibandingkan MQTT *broker* dengan basis *image* belenalib/raspberry-pi2-alpine, dan MQTT *broker* yang dijalankan secara *native* dalam hal *packet loss*. Selain itu fitur QoS pada MQTT *broker* tetap dapat berkerja dengan baik walaupun, dijalankan diatas docker.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan pada penelitian yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut:

1. Pengujian MQTT *Broker* berbasis kontainer menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan belenalib/raspberry-pi2-alpine pada perangkat Raspberry Pi dilakukan dengan 3 parameter pengujian yaitu pengujian *resource usage* (*cpu* dan *memory usage*), pengujian *latency* dan *pengujian packet loss.* Pengujian dilakukan dengan menambahkan variasi jumlah klien *publisher/subsriber* mulai dari 200, 400, 600, 800, 1000 dengan perbandingan 50% *publisher* dan 50% *subscriber*.

2. Kinerja MQTT *Broker* mengunakan docker dengan *basis image* belenalib/raspberry-pi2 atau Raspbian dan MQTT *Broker* tanpa mengunakan docker atau *native* memiliki peformansi yang seimbang namun memiliki kekuatan pada point pengujian yang berbeda. Dimana MQTT *Broker* dengan *basis image* belenalib/raspberry-pi2 memiliki kinerja yang baik pada pengujian CPU *usage* dan *packet loss*. Hal ini terbukti dengan hanya menggunakan terendah 16,30% CPU dan tertinggi 99,86% CPU, kemudian pada pengujian *packet loss* hanya mengalami *packet loss* 28,60% pada QoS 0. Sedangkan MQTT *Broker* tanpa mengunakan docker atau *native* memiliki kinerja yang baik pada pengujian *memory usage* dan *latency*. Hal ini terbukti dengan hanya menggunakan *memory* 3,9 MB (terendah) dan 8,18 MB (tertinggi). Lalu pada pengujian *latency* hanya 0.08 sampai 0.5. Bisa dikatakan sangat jauh dibandingkan MQTT *Broker* mengunakan docker baik belenalib/raspberry-pi2 apalagi belenalib/raspberry-pi2-alpine. MQTT *Broker* berbasis kontainer menggunakan Docker dengan basis *image* belenalib/raspberry-pi2-alpine memiliki kinerja yang terendah daripada MQTT *Broker* berbasis kontainer menggunakan Docker dengan basis *image* belenalib/raspberry-pi2 dan MQTT *Broker* tanpa mengunakan docker. Namun basis *image* ini memiliki ukuran yang jauh lebih kecil sebersar 21,5MB di bandingkan dengan basis *image* belenalib/raspberry-pi2 yang memiliki ukuran *image* bahkan sampai 187MB.

6.2 Saran

Beberapa saran untuk penetilian lebih lanjut yaitu:

1. Menguji MQTT *broker* menggunakan Docker dengan parameter lain.
2. Menguji MQTT *broker* menggunakan Docker pada perangkat lain seperti Raspberry Pi Zero.

DAFTAR REFERENSI

Armitage, G., Claypoll, M., Branch, P. (2006). Networking and Online Games: Understanding and Engineering Multiplayer Internet Games. Chichester: John Wiley & Sons, Ltd.

Ashton, K. (2009). That "Internet of Things" Thing: In the Real World Things Matter More than Ideas. RFID Journal. [online] Tersedia di http://www.rfidjournal.com/articles/view?4986 [Diakses 1 September 2018].

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 1(3), 81–84.

Chelladhurai, J. S., Singh, V., dan Raj, P. (2017). Learning Docker Second Edition. Birmingham: Packt Publishing Ltd.

Coulouris, G., Dollimore, J., Kindberg, T., dan Blair, G. (2012). DISTRIBUTED SYSTEMS: Concepts and Design Fifth Edition. Boston: Addison-Wesley.

Docker Inc. (-). Docker Overview. [online] Docker. Tersedia di: https://docs.docker.com/engine/docker-overview/ [Diakses: 1 September 2018].

Fysarakis, K., Askoxylakis, I., Soultatos, O., Papaefstathiou, I., Manifavas, C. dan Katos, V. (2016). Which IoT Protocol? Comparing Standardized Approaches over a Common M2M Application. 2016 IEEE Global Communications Conference (GLOBECOM).

HiveMQ. (2015). MQTT Essential Part 3: Client, Broker and Connection Establishment. [online] HiveMQ. Tersedia di: HiveMQ: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/ [diakses 7 Desember 2018].

Hui, J. (2018). MQTT broker latency measure tool. [Perangkat Lunak]. Tersedia di: https://github.com/hui6075/mqtt-bm-latency [diakses 7 Desember 2018].

International Telecomunication Union (2005). The Internet of Things. ITU Internet Report 2005.

Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol, The Journal of Open Source Software, vol. 2, no. 13.

Linux Academy. (2017). Containers for Everyone. [e-book] Linux Academy. Tersedia di: https://linuxacademy.com/templates/default/assets/pdf/containers-for-everyone-ebook.pdf [Diakses 12 Februari 2018]

Loukides, M., dan Bruner, J. (2015). What is Internet of Things?. Sebastopol: O'Reilly Media, Inc.

Miller, L. (2017). Internet of Things For Dummies, Qorvo Special. Hoboken: John Wiley & Sons, Inc.

Morabito, R. (2016). A performance evaluation of container technologies on Internet of Things devices. 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).

Mqtt.org. (-). FAQ - Frequently Asked Questions | MQTT. [online] MQTT. Tersedia di: http://mqtt.org/faq [Diakses 1 September 2018].

Nicholson, J. (2014). What is high system resource usage?. Inmotion Hosting. Tersedia di: http://www.inmotionhosting.com/support/website/server-usage/what-is-high-system-resource-usage [diakses 31 Januari 2019]

OASIS. (2014). MQTT Version 3.1.1. [online] Tersedia di: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718027 [Diakses 1 Oktober 2018].

Przyłucki, S., Czerwiński, D., dan Sierszeń, A. (2017). A performance evaluation of docker-based MQTT server implementation on internet of things device. Studia Informatica, 38(3), 89-99.

Raspberry Pi Foundation. (-). Raspberry Pi — Teach, Learn, and Make with Raspberry Pi. [online] Raspberry Pi. Tersedia di: https://www.raspberrypi.org/ [Diakses 1 Sep. 2018].

Van Steen, M., dan Tanenbaum, A. S. (2017). *Distributed systems*. Leiden, The Netherlands: Maarten van Steen.

Suresh, P., Daniel, J., Parthasarathy, V. dan Aswathy, R. (2014). A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. 2014 International Conference on Science Engineering and Management Research (ICSEMR).

Thulin, M. (2004). Measuring Availability in Telecommunications Networks. Stockholm.

Vemulapalli, R., dan Mada, R. K. (2014). “Performance of Disk I/O operations during the Live Migration of a Virtual Machine over WAN,”

VMWare. (2007). “Understanding Full Virtualization, Paravirtualization, and Hardware Assist.” [e-book]. Tersedia di: http://www.vmware.com/techpapers/2007/understanding-full-virtualizationparavirtualizat-1008.html. [Diakses: 17 Februari 2018].